Pedro Kantek

Inteligência Artificial

Unicenp
Set/2007
Curitiba

 ©88-09, Pedro Kantek

Sumário

1 Questões Operacionais **5**
- 1.1 Regras da disciplina . . . 5
  - 1.1.1 Portal . . . 5
  - 1.1.2 Exercícios Individuais . . . 5
- 1.2 Uma taxonomia para a IA . . . 9
- 1.3 Bibliografia para Inteligência Artificial . . . 11

2 uma visão histórica da IA **15**
- 2.1 Máquina de Turing . . . 15
- 2.2 Teorema de Goedel . . . 25
  - 2.2.1 Desafio . . . 27

3 Uma visão moderna da IA **43**
- 3.1 Introdução . . . 43
  - 3.1.1 Inteligência Artificial . . . 43
  - 3.1.2 Do que se ocupa a IA . . . 44
  - 3.1.3 O que é inteligente ? . . . 44
  - 3.1.4 Captchas . . . 45
  - 3.1.5 Enfoques do pensamento . . . 47
- 3.2 História da IA . . . 48
  - 3.2.1 Agentes . . . 48
  - 3.2.2 As ciências necessárias . . . 49
  - 3.2.3 O começo . . . 49
  - 3.2.4 Futuro . . . 51
  - 3.2.5 Exercício . . . 51
- 3.3 Sistemas Baseados em Conhecimento . . . 53
- 3.4 Agentes . . . 53
  - 3.4.1 Mapeamento entre seqüência de percepção e ações . . . 54
  - 3.4.2 Agente como um programa . . . 54
  - 3.4.3 Algoritmo básico de um agente . . . 56
- 3.5 Agentes inteligentes . . . 56
  - 3.5.1 Propriedades do ambiente . . . 58
  - 3.5.2 O mundo dos aspiradores de pó . . . 59
  - 3.5.3 Exercícios . . . 61

4 Busca em espaço de estados **65**
- 4.1 Introdução . . . 65
  - 4.1.1 Definição de problema . . . 65
  - 4.1.2 Estratégias de busca . . . 70
- 4.2 Métodos inteligentes de busca . . . 74
  - 4.2.1 Exercício . . . 76

4.2.2 A* . . . 77
4.2.3 Quebra-cabeça 8 . . . 77
4.2.4 Solução do cubo RUBIK . . . 78
4.2.5 Resumo . . . 79
4.3 A*, continuação . . . 80
4.3.1 Mais sobre a heurística . . . 80
4.3.2 como inventar funções heurísticas . . . 80
4.3.3 O papel de k_1 e k_2 . . . 81
4.3.4 Melhoramento interativo . . . 82
4.4 JOGOS . . . 87
4.4.1 Tipos de jogos . . . 89
4.4.2 Algoritmo Básico . . . 89
4.4.3 Algoritmo minimax . . . 89
4.4.4 Corte α - β . . . 91
4.4.5 Algoritmo $\alpha - \beta$ . . . 93
4.5 O jogo do xadrez . . . 94
4.5.1 O tabuleiro . . . 94
4.5.2 As peças . . . 95
4.5.3 O jogo . . . 95
4.5.4 Notação . . . 98
4.6 Jogadores automáticos de xadrez . . . 101
4.6.1 A Sexta partida Deep Blue X Kasparov . . . 103
4.6.2 Idéias para uma implementação . . . 114
4.6.3 Função de avaliação . . . 114
4.6.4 Gerador de movimentos . . . 115
4.6.5 Construtor da árvore e implementador do $\alpha - \beta$ . . . 115
4.6.6 Trabalho pedido para as turmas de IA em 2004 . . . 115

5 Redes Neurais **121**
5.1 Introdução . . . 121
5.1.1 Programa convencional versus RNA . . . 121
5.1.2 O neurônio . . . 122
5.1.3 Um neurônio artificial . . . 123
5.1.4 Funções de ativação . . . 124
5.1.5 Arquiteturas de RNA . . . 125
5.1.6 ALVINN . . . 125
5.1.7 NetTalk . . . 127
5.2 Um exemplo prático . . . 128
5.2.1 O aprendizado da rede . . . 130
5.2.2 Algoritmo de retropropagação (backpropagation) . . . 131
5.2.3 Um exemplo real . . . 131
5.2.4 Fenômeno do super-aprendizado . . . 133

6 Computação Evolutiva **135**
6.1 Introdução . . . 135
6.1.1 Resolvendo problemas . . . 135
6.2 Genética . . . 136
6.2.1 Gregor Mendel . . . 136
6.2.2 DNA . . . 137
6.2.3 Evolução . . . 138
6.3 Algoritmos genéticos . . . 139
6.4 Vocação . . . 139

6.4.1 Conceituação . . . 140
6.5 Implementação . . . 146
6.5.1 Técnicas de Otimização . . . 147
6.5.2 Formalização algorítmica . . . 148
6.5.3 Formalização matemática . . . 149
6.5.4 Terminologia . . . 152
6.5.5 Idéias Comuns à CE . . . 152
6.6 CE: mais conceitos . . . 154
6.6.1 Hibridização . . . 154
6.6.2 Roda da roleta . . . 155
6.6.3 Mutação de bit . . . 155
6.6.4 Normalização linear . . . 156
6.6.5 Janelamento . . . 156
6.6.6 Recombinação de 2 pontos . . . 156
6.6.7 Recombinação uniforme . . . 157
6.6.8 Trabalho pedido em 2002 . . . 158
6.7 Programação Genética . . . 162
6.7.1 Outros ramos da Computação Evolutiva . . . 162
6.7.2 Evoluindo programas (e não dados) . . . 163
6.7.3 Um exemplo pedagógico . . . 164
6.7.4 Evoluindo um robot . . . 165
6.7.5 O processo . . . 167
6.7.6 Um caso real . . . 168
6.7.7 Implementação . . . 170
6.8 Orientações para uso do GALOPPS . . . 172
6.9 TUDEL . . . 175
6.9.1 Funcionalidades do tudel . . . 176
6.9.2 Operação do tudel . . . 178
6.9.3 Exemplo real do tudel . . . 180
6.9.4 Funções APL desenvolvidas . . . 180
6.9.5 Estudo de caso . . . 182
6.9.6 Análise de normalizadores e seletores . . . 184
6.9.7 Análise de taxas . . . 185
6.9.8 Análise de tamanhos . . . 185
6.9.9 Repetição de resultados . . . 186
6.10 Algumas aplicações de Algoritmos Genéticos . . . 186
6.10.1 Aplicações de interesse histórico . . . 187

7 LISP **197**
7.1 Introdução . . . 197
7.1.1 Programação Funcional . . . 197
7.1.2 História do LISP . . . 199
7.1.3 Como obter um LISP . . . 200
7.1.4 Introdução e conceitos iniciais . . . 200
7.1.5 Funções . . . 201
7.1.6 Símbolos e Números . . . 201
7.1.7 Listas . . . 207
7.2 Funções CAR e CDR . . . 212
7.2.1 Representação de funções LISP . . . 213
7.2.2 Definição de funções . . . 214
7.2.3 Usando um computador . . . 217
7.2.4 Aritmética . . . 220

 ©88-09, Pedro Kantek

7.3 Formas condicionais . . . 222
7.3.1 Função APPLY . . . 233
7.3.2 Variáveis globais e locais . . . 233
7.3.3 Impressão . . . 234
7.3.4 Função LET . . . 235
7.3.5 Interação . . . 236
7.4 Árvores: Representação com ponto . . . 238
7.5 Operadores aplicativos . . . 241
7.5.1 Notação LAMBDA . . . 242
7.5.2 Classificação . . . 244
7.5.3 Algumas funções interessantes . . . 244
7.5.4 Entrada e Saída em LISP . . . 244
7.5.5 Para você fazer . . . 247
7.5.6 Desafios . . . 248
7.6 MYCIN . . . 248
7.7 Aplicação prática . . . 249
7.7.1 Um tradutor simples . . . 249
7.7.2 Um censurador de texto . . . 251

8 Sistemas de Produção 253
8.1 Introdução . . . 253
8.1.1 Tarkin revisitado . . . 255
8.2 DENDRAL . . . 256
8.2.1 Como funciona . . . 257
8.2.2 MYCIN . . . 258
8.3 Fatores de certeza . . . 261
8.3.1 Outros . . . 261
8.4 Um *shell* para sistemas especialistas; ExSinta . . . 262
8.4.1 Arquitetura . . . 262
8.4.2 Formato das regras . . . 263
8.4.3 Fatores de confiança . . . 264
8.4.4 Implementação . . . 264
8.5 Implementação de um SBC . . . 265
8.5.1 Representação, Raciocínio e Lógica . . . 266

9 Programação Lógica 269
9.1 Histórico . . . 269
9.1.1 Cláusulas de Horn . . . 269
9.1.2 Conceitos básicos . . . 270
9.1.3 Introdução . . . 272
9.1.4 Programas . . . 272
9.1.5 Cálculos . . . 276
9.1.6 Dados . . . 279
9.2 Um tutorial . . . 280
9.3 Lógica de Primeira Ordem . . . 280
9.3.1 Síntaxe . . . 281
9.3.2 Mais exemplos . . . 285
9.3.3 Exercícios do [Rus96] . . . 285
9.4 Inferência na LPO . . . 286
9.4.1 Exemplo de uma demonstração . . . 287
9.4.2 Resolução × refutação . . . 292
9.4.3 Em Resumo . . . 292

9.4.4 Exemplo . . . 296
9.4.5 Exercícios . . . 297
9.4.6 Resolução na LPO . . . 298
9.4.7 Exemplo . . . 298
9.5 Apêndice . . . 299
9.5.1 Unificação em LISP . . . 299

10 Como representar conhecimento 303
10.0.2 Redes Semânticas . . . 304
10.0.3 Quadros . . . 308
10.0.4 Grafos Conceituais de Sowa, 1984 . . . 311
10.0.5 Roteiros . . . 312
10.1 FASE . . . 312
10.1.1 Manipulador de regras . . . 312
10.1.2 Resolutor de conflitos . . . 313
10.1.3 Representador de conhecimentos . . . 313
10.1.4 Controlador de estratégia . . . 314
10.1.5 Forma de uso . . . 314
10.1.6 Tutorial no uso do FASE (UMA PARTE DO FASE...) . . . 318

11 Visão 325
11.1 Cérebro . . . 327
11.1.1 Rodopsina . . . 328
11.1.2 Uma visão bioquímica do ciclo . . . 328
11.2 Psicologia da visão . . . 329
11.2.1 Inteligência Visual . . . 330
11.2.2 Ilusões de ótica . . . 331
11.3 Estereogramas . . . 331
11.4 Tomografia computadorizada axial . . . 333
11.4.1 Outras técnicas de reconstituição por imagem . . . 341
11.5 imagens mapeadas . . . 341
11.6 Arquivos BMP . . . 343
11.7 Modificação de histograma . . . 347
11.8 Escalonamento de histograma . . . 348
11.8.1 Equalização de histograma . . . 348
11.8.2 Distribuição linear de níveis de cinza . . . 349
11.8.3 Compressão de histograma . . . 350
11.8.4 Limiarização . . . 350
11.9 Manipulações aritméticas . . . 351
11.9.1 Negativização . . . 352
11.9.2 Multiplicação por constante k . . . 353
11.9.3 Adição e subtração de 2 imagens . . . 353
11.9.4 E e OU lógicos em imagens binárias . . . 354
11.9.5 Zoom . . . 354
11.10 Filtragem, realce e suavização . . . 355
11.10.1 Filtro da média e da mediana . . . 355
11.10.2 Espalhamento de tinta . . . 357
11.10.3 Componentes Conectados . . . 358
11.10.4 Trabalhos de anos passados . . . 360
11.11 Operadores Morfológicos . . . 364
11.11.1 Erosão Binária . . . 364
11.11.2 Dilatação Binária . . . 367

 ©88-09, Pedro Kantek

11.11.3 Detecção de Bordas . . . 368
11.11.4 Abertura Binária . . . 369
11.11.5 Fechamento Binário . . . 370
11.11.6 Transformação hit-miss . . . 371
11.11.7 Afinamento . . . 372
11.11.8 Espessamento . . . 375
11.11.9 Esqueletização . . . 375
11.12 Segmentação por contorno ativo . . . 375
11.13 Uma aplicação real: AFIS . . . 376
11.14 Análise de Imagens . . . 379
11.14.1 exemplo: reconhecendo placas . . . 379
11.14.2 Mundo dos cubos . . . 380
11.15 Exercícios . . . 383

12 Robótica **391**
12.0.1 Sensores . . . 393
12.0.2 Efetuadores . . . 393
12.1 Grand Challenge . . . 393
12.1.1 Regras . . . 394
12.1.2 O percurso . . . 394
12.1.3 O algoritmo do sandstorm . . . 394
12.1.4 Versão 04 . . . 395
12.1.5 Versão 05 . . . 396
12.2 IBM7535 . . . 398
12.2.1 Área de trabalho . . . 399
12.2.2 Comandos . . . 401
12.2.3 Alguns exemplos . . . 403
12.2.4 Para treinar . . . 404

13 Processamento em Linguagem Natural **405**
13.0.5 Formalismos gramaticais . . . 405
13.0.6 Máquina de estados . . . 407
13.0.7 Analisador Recursivo Descendente . . . 409
13.0.8 Analisador por remoção de ruído . . . 412

14 Aprendizado **415**
14.0.9 Aprendizado simbólico . . . 415
14.0.10 Busca em espaço de versões . . . 416
14.0.11 Algoritmo de eliminação de candidatos . . . 417
14.1 Qual a idade dos meus 3 filhos ? . . . 418
14.2 Alguns conceitos caros à IA . . . 419
14.2.1 A volta de Santa Catarina . . . 419
14.2.2 Tentativa de solução . . . 420
14.2.3 Diversos métodos de solução . . . 420
14.2.4 Diagramas e esquemas: a árvore . . . 426

...load your program
I am yourself...
I am perfect,
and are You ?

Karn Evil-9 3^{rd} impression
do LP (depois CD 719124-2) Brain, Salad and Surgery
Emerson, Lake & Palmer

Saber se os computadores pensam é tão importante como saber se os submarinos nadam.

(Edsger Dijkstra, citado por Pedro Guerreiro)

 ©88-09, Pedro Kantek

Capítulo 1

Questões Operacionais

1.1 Regras da disciplina

Estas são as regras propostas para a cadeira de Inteligência Artificial no corrente ano.

1. Em todas as aulas haverá um exercício prático individual e diferente sobre o assunto da aula que será dada.
2. O exercício em questão deverá ser devolvido até uma data previamente estabelecida, sob pena de multa de 50% da nota devida
3. Em cada aula poderá haver a devolução pelo professor para o aluno do exercício anterior devidamente corrigido. Eventuais discussões quanto à nota lançada e formas de solução do exercício também poderão ter lugar aqui.
4. Não haverá segunda chamada para exercícios perdidos.
5. Não serão aceitos xerox, fax ou e-mail de exercícios. Apenas o exercício original que contiver o nome do aluno será aceito.
6. A nota bimestral será obtida fazendo a média aritmética dos exercícios do bimestre, com valor de 60%, junto com a nota da prova que valerá 40%..
7. Na aula seguinte à da prova o professor distribuirá um extrato no qual constarão as notas obtidas por cada aluno em cada exercício individual.
8. Exercícios que exijam implementação poderão ter um prazo de entrega maior, dependendo de combinação a ser feita em aula.

1.1.1 Portal

No Portal Unicenp (www.unicenp.edu.br) estão os textos base de todas as aulas da disciplina. na forma de um livro de nome LIVROIA.PDF de autoria de P. Kantek.

1.1.2 Exercícios Individuais

código

São gerados usando um pacote denominado Algoritmos Vivos, que está em desenvolvimento desde o ano de 1992, quando começou a ser usado ainda na Universidade Católica do Paraná, para a disciplina de estrutura de dados. Hoje é uma coleção de mais de 210 exercícios abrangendo as áreas de algoritmos, estruturas de dados, tópicos avançados

em informática e inteligência artificial. Cada exercício possui um código que o remete a uma classificação de temas dentro da Ciência da Computação, e uma seqüência que informa qual o posição do exercício dentro do ano letivo.

Por exemplo, o exercício a ser feito hoje, tem o código VIVO960a (VIVO é a identificação do pacote gerador de exercícios, o número 96x remete problemas específicos para IA e a letra "a" indica o primeiro programa dentro do workspace).

Seqüência

cada exercício tem um identificador único, chamado seqüência com as seguintes informações: **aaIAM101**, que deve ser assim interpretada:

aa Ano atual

IA disciplina de IA

M ou N Turno

1 1^o bimestre do ano letivo

01 Primeiro exercício deste bimestre

O autor, pelo autor

Meu nome é Pedro Luis Kantek Garcia Navarro, conhecido como Kantek, ou Pedro Kantek. Nasci em Curitiba há mais de 50 anos. Sou portanto brasileiro, curitibano e coxa-branca com muito orgulho, mas sendo filho de espanhóis (meus 7 irmãos nasceram lá), tenho também a nacionalidade espanhola. Aprendi a falar em *castellano*, o português é portanto meu segundo idioma. Estudei no Colégio Bom Jesus e quando chegou a hora de escolher a profissão, lá por 1972, fui para a engenharia civil, mas sem muita convicção. Durante a copa do mundo de futebol de 1974 na Alemanha, ao folhear a Gazeta do Povo, achei um pequeno anúncio sobre um estágio na área de processamento de dados (os nomes informática e computação ainda não existiam). Lá fui eu para um estágio na CELEPAR, que hoje, mais de 30 anos após, ainda não acabou. Na CELEPAR já fui de tudo: programador, analista, suporte a BD (banco de dados), suporte a TP (teleprocessamento), coordenador de auto-serviço, coordenador de atendimento, ... Atualmente estou na área de governo eletrônico. Desde cedo encasquetei que uma boa maneira de me obrigar a continuar estudando a vida toda era virar professor. Comecei essa desafiante carreira em 1976, dando aula num lugar chamado UUTT, que não existe mais. Passei por FAE, PUC e cheguei às Faculdades Positivo em 1988. Sou o decano do curso de informática e um dos mais antigos professores da casa. Na década de 80, virei instrutor itinerante de uma empresa chamada CTIS de Brasília, e dei um monte de cursos por este Brasil afora (Manaus, Recife, Brasília, Rio, São Paulo, Fortaleza, Floripa, ...). Em 90, resolvi voltar a estudar e fui fazer o mestrado em informática industrial no CEFET. Ainda peguei a última leva dos professores franceses que iniciaram o curso. Em 93 virei mestre, e a minha dissertação foi publicada em livro pela editora Campus (Downsizing de sistemas de Informação. Rio de Janeiro: Campus, 1994. 240p, ISBN:85-7001-926-2). O primeiro cheque dos direitos autorais me manteve um mês em Nova Iorque, estudando inglês. Aliás, foi o quarto livro de minha carreira de escritor, antes já havia 3 outros (MS WORD - Guia do Usuário Brasileiro. Rio de Janeiro: Campus, 1987. 250p, ISBN:85-7001-507-0, Centro de Informações. Rio de Janeiro: Campus, 1985. 103p, ISBN:85-7001-383-3 e APL - Uma linguagem de programação. Curitiba. CELEPAR, 1982. 222p). Depois vieram outros. Terminando o mestrado, rapidamente para não perder o fôlego, engatei o doutorado em engenharia elétrica. Ele se iniciou em 1994 na UFSC em Florianópolis. Só terminou em 2000, foram 6 anos inesquecíveis, até porque nesse meio tive que aprender o francês - mais um mês em Paris aprendendo-o. Finalmente virei engenheiro, 25 anos depois de ter iniciado a engenharia civil. Esqueci de dizer que no meio do curso de Civil desistí (cá pra nós o assunto era meio chato...) em favor de algo muito mais emocionante: matemática. Nessa época ainda não havia cursos superiores de informática. Formei-me em matemática na PUC/Pr em 1981. Em 2003, habilitei-me a avaliador de cursos para o MEC. Para minha surpresa, fui selecionado e virei delegado do INEP (Instituto Nacional de Pesquisas Educacionais) do Governo Brasileiro. De novo, visitei lugares deste Brasilzão que sequer imaginava existirem (por exemplo, Rondonópolis, Luiziânia, Rio Grande, entre outros), sempre avaliando os cursos na área de informática: sistemas de informação, engenharia e ciência da computação. Atualmente estou licenciado da PUC e no UNICENP respondo por 4 cadeiras: Algoritmos (1. ano de sistemas de informação), Estrutura de Dados (2.ano, idem) e Tópicos Avançados em Sistemas de Informação (4.ano, idem), além de Inteligência Artificial (último ano de Engenharia da Computação). Já fiz um bocado de coisas na vida, mas acho que um dos meus sonhos é um dia ser professor de matemática para crianças: tentar despertá-las para este mundo fantástico, do qual – lastimavelmente – boa parte delas nunca chega sequer perto ao longo de sua vida.

 ©88-09, Pedro Kantek

O autor, há muitos anos, quando ainda tinha abundante cabeleira

1.2 Uma taxonomia para a IA

- Definições, localização conceitual da IA e História da IA
- Complexidade Computacional, Explosão combinatória
- Problemas e espaços de estados. Técnicas de busca
  - Busca em largura
  - Busca em profundidade
  - Algoritmo A*
  - Jogos, grafos E/OU
    - Algoritmo Minimax
    - Poda $\alpha - \beta$
- Modelos de programação. Programação funcional. Programação lógica
  - LISP
    - Tipos de dados; Listas e demais estruturas; Controle de execução e recursão; Entrada e saída; Orientação a objetos: CLOS
  - PROLOG
    - Predicados; Variáveis e unificação; Regras; Busca e Recursão; Listas
- Representação do conhecimento
  - Lógica de Predicados
  - Representação através de regras
    - Tratamento da incerteza
  - Representação através de redes semânticas
  - Representação com frames
  - Representação através de scripts: senso comum
- Raciocínio automatizado
  - Cálculo de predicados
  - Regras de inferência
    - Unificação
    - Resolução
    - Universo e Base de Herbrand
  - Raciocínio baseado em casos
  - Lógica nebulosa (fuzzy)
  - Raciocínio não monotônico
- Aprendizagem e sistemas adaptativos
  - Motivação, objetivos, definições e avaliação de desempenho
  - Métodos simbólicos e estatísticos
  - Redes Neurais
    - Redes neurais naturais
    - Natureza da computação neural

 ©88-09, Pedro Kantek

        * Perceptrons
        * Retroprogação
        * Redes de Hopfield e memória associativa
        * aprendizagem sem supervisão
    - Computação Evolutiva
        * Algoritmos genéticos
        * Estratégia evolutiva
        * Programação evolutiva

- Percepção e reconhecimento
    - Visão
        * Aquisição de imagens
        * Processamento da imagem
        * Segmentação
        * Descrição e interpretação
        * Morfologia matemática
        * Visão tridimensional e tratamento do movimento
    - Reconhecimento da fala

- Sistemas Baseados em Conhecimento (antigos SEs)
    - Definição, motivação, origem
    - Arquiteturas de um SBC. Critérios de aplicação de um SBC. Ciclo de vida
    - Conhecimento e sua representação: lógica, redes semânticas, frames, scripts...
    - Raciocínio e inferência
    - Engenharia do conhecimento
    - Aquisição do conhecimento
    - Validação e verificação de SBCs

- Processamento de linguagem natural
    - Bases da linguística
    - Síntaxes e gramáticas generativas
    - Representações de semântica
    - Problema da ambigüidade
    - Gerenciamento de diálogos
    - enfoques da tradução automatizada

- Inteligência artificial distribuída
    - Agentes distribuídos
    - inteligência de colméia

- Exploração de Dados (data mining)
    - Introdução, métodos, resultados

- Robótica
    - Introdução, métodos, resultados

1.3 Bibliografia para Inteligência Artificial

Esc03 ESCOLANO RUIZ, Francisco et alli. Inteligencia Artificial. Ed. Thompson, Madrid.

Nil03 NILSSON, Nils. Artificial Intelligence: a New Synthesis. (Unicenp: 006.3 N712a, 1 ex.)

O último livro (uma belíssima síntese de toda a obra) do pesquisador tradicional Nils Nilsson.

Rus95 RUSSEL, Stuart e NORVIG, Peter. Artificial Intelligence - a modern approach. Prentice Hall, 1995. (Unicenp: 006.3.R967a, 3ex.)

O livro usado em 9 de cada 10 boas universidades no mundo na área de IA

Rus96 RUSSEL, Stuart e NORVIG, Peter. Inteligencia Artificial - un enfoque moderno. Pearson Educacion, 1996.

O mesmo livro acima, traduzido (mal) para o espanhol.

Rus03 RUSSEL, Stuart e NORVIG, Peter. Inteligencia Artificial. Editora Campus, 2003. (Unicenp: 006.3 R967i, 3ex.)

O livro acima, na sua segunda edição, traduzido para o português.

Lug02 LUGER, George F. Inteligência Artificial. Editora Bookman. 2002.

Tradução desta obra que é bastante antiga e tradicional para o português.

Dew89 DEWDNEY, A.K. The (new) Turing Omnibus. W.H.Freeman, 2001.

Um livro iluminado sobre toda a ciência da computação.

Ata99 ATTALAH, Mikhail. (org). Algorithms and Theory of Computation Handbook. CRC Press. (Unicenp: 005.12 A396al, 1 ex.)

Talvez o mais completo e moderno livro de algoritmos.

Bar81 BARR, Avron e FEIGENBAUN, Edward. The Handbook of Artificial Inteligence. Vols 1-3. Addison Wesley, 1981.

O estado da arte em 1981.

Kvi88 KVITCA, Adolfo. Resolucion de problemas con inteligencia artificial. Edição EBAI, 1988.

Apresenta bem o tema heurística.

Mic85 MICHIE, Donald e JOHNSTON, Rory. O computador Criativo. Editoral Presença, Lisboa, 1985. (Unicenp: 004.M624c, 1 ex.)

Uma boa introdução à filosofia da IA.

Ric94 RICH, Elaine e KNIGHT Kevin. Inteligência artificial. Makron Books do Brasil. 1994. (Unicenp: 006.3 R498e, 15 ex.)

Tradução (horrível) do original americano. Razoável introdução.

Bit99 BITTENCOURT, Guilherme. Inteligência Artificial - ferramentas e teoria. UFSC, 1999.

Bom texto introdutório, principalmente sobre lógica.

 ©88-09, Pedro Kantek

Win84 - WINSTON, Patrick Henry. Artificial Intelligence. Addison-Wesley. 1984.

O livro é meio confuso, mas enfim...

Sch89 SCHILDT, Herbert. Inteligência Artificial Utilizando a Linguagem C. McGraw-Hill. 1989.

Um livro com exemplos bem simples, simplórios até, mas todos programados em C.

Gol89 GOLDBERG, David. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, 1984. (Unicenp: 006.31 G618g, 1ex.)

O livro seminal em algoritmos genéticos.

Dav91 DAVIS, L. Handbook of genetic algorithms. Van Nostrand Reinhold. 1991. (Unicenp: 006.31 H236h, 1 ex.)

Um bom e didático livro sobre GA.

Loe96 LOESCH, Claudio e SARI, Solange. Redes Neurais Artificiais - Fundamentos e Modelos. Editora FURB. 1996.

Básico sobre redes neurais. Meio confuso.

Kov97 KOVACS, Z. L. Redes Neurais Artificiais - Fundamentos e aplicações. Editado pelo autor. 1997

Pouco ambicioso.

Kov97b KOVACS, Z.L. O cérebro e a sua mente. Introdução à neurociência computacional. Edição do Autor. 1997.

Boa introdução à neurociência.

Hof00 HOFFMAN, Donald. Inteligência Visual - como criamos o que vemos. Editora Campus. 2000. (Unicenp: 159.937 H699i, 1 ex.)

Excelente livro sobre o (pouco que conhecemos do) funcionamento da parte visual do cérebro.

Wei92 WEIZENBAUM, Joseph. O poder do computador e a razão humana. Edições 70. 1992.

Do criador do ELIZA, um bom texto filosófico e introdutório.

Pen91 PENROSE, Roger. A Mente nova do rei. Computadores, mentes e as leis da física. Ed. Campus, 1991. (Unicenp: 006.3 P417m, 1 ex.)

Livro indigesto, mas cheio de pérolas.

Rab95 RABUSKE, Renato. Inteligência Artificial. UFSC 1995.

Pouco ambicioso.

Fis87 FISCHLER, Martin. Intelligence - The eye, the brain and the computer. Addison-Wesley, 1987.

Mais questões sobre a visão, o cérebro e a inteligência.

Cre93 CREVIER, Daniel - AI the tumultuous history of the search for artificial intelligence. Harper Collins. 1993

A história da IA.

Fri87 FRIANT, Jean. Juegos lógicos en el mundo de inteligencia artificial. Gedisa. 1987.

alguns jogos e problemas a resolver usando IA

Bar03 BARONI, Dante et alli. Sociedades Artificiais - a nova fronteira da inteligência das máquinas. Bookman. 2003. (Unicenp: 006.3 B265s, 2 ex.)

Diversos artigos sobre o tema IA.

Rob64 ROBERT, Marthe. A revolução psicanalítica. Moraes Editores, Lisboa. 1964.

Uma leve introdução à psicanálise.

Lev93 LEVY, Steven. Vida Artificial - em demanda de uma nova criação. Dom Quixote. Lisboa, 1993

Uma análise interessante sobre o tópico vida artificial.

Sag92 SAGAN, Carl. Os Dragões do Eden. 1992.

Um belo livro sobre a estrutura e o funcionamento do cérebro animal.

Str97 STRATHERN, Paul. Turing e o computador em 90 minutos. Jorge Zahar Editora. (Unicenp: 004.S899t = 4.andar).

 ©88-09, Pedro Kantek

Capítulo 2

uma visão histórica da IA

2.1 Máquina de Turing

Embora a humanidade use o conceito de algoritmo há milênios (O de MDC é devido a Euclides), e o próprio nome algoritmo derive do autor árabe de um tratado de álgebra (por volta de 800 dC), a definição precisa do que seja um algoritmo geral é apenas de 1930. Ela é devida a Turing, através do seu conceito de Máquina de Turing. Esta não é uma máquina real, sendo apenas uma abstração matemática, daí o seu poder de encanto.

No final do século XIX assistia-se a um esforço de sistematização da matemática. Conceitos até então vagos (como infinitesimal, ou conjunto, ...) sofriam esforços por parte dos matemática em formalização e definição rigorosa. Em 1900, o matemático alemão Hilbert apresentou em um congresso uma lista de problemas que segundo ele *estariam ocupando as mentes matemáticas no século que ora se iniciava.* O décimo problema de Hilbert, dizia: **...haverá algum procedimento mecânico geral que possa em princípio resolver todos os problemas da matemática ?** [1]

Em 1937, Turing respondeu a esta pergunta através do conceito de Máquina de Turing. Ela é composta por um número finito de estados (ainda que possa ser muito grande). Deve tratar um input infinito. Tem um espaço exterior de armazenagem também infinito. Turing imaginou o espaço exterior para dados e armazenamento como sendo uma fita, que pode mover-se para frente e para trás. Além disso a máquina pode ler e escrever nesta fita. A fita está composta por quadrados (ou seja, o nosso ambiente é discreto). Apenas 2 símbolos podem ser escritos nos quadrados da fita, digamos 0 (quadrado em branco) e 1 (quadrado preenchido), ou vice-versa.

Isto posto, lembremos que:

1. os estados internos do aparelho são finitos em número
2. o comportamento da máquina é totalmente determinado pelo seu estado interior e pelo input.

Ou seja, dado o estado inicial, e um símbolo de input, o aparelho deve ser determinista. Logo ele,

- muda seu estado interno para outro, e/ou
- Muda o simbolo que foi lido pelo mesmo ou por outro, e/ou

[1] Eis o problema original, como proposto por Hilbert: Determination of the solvability of a diophantine equation: Given a diophantine equation with any number of unkown quantities and with rational numerical coefficients: to devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers

- movimenta-se para a direita ou para a esquerda, 1 quadrado, e/ou
- Decide se continua ou interrompe e para.

Definição de uma máquina de Turing Para especificarmos nossa máquina de Turing, teríamos que escrever algo do tipo

Estado atual	Sinal lido	faz o que	Vai para estado	Escreve	Anda para onde
0	0	→	0	0	D
0	1	→	13	1	E
1	0	→	65	1	D
1	1	→	1	0	D
2	0	→	0	1	D.PARE
2	1	→	66	1	E

Esta definição acima, é adequada para a tarefa de numeração das Máquinas de Turing, fenômeno esse que é central na teoria. Entretanto, para operar com esta máquina, talvez seja mais adequado escrever uma tabela como

estado	chega	escreve?	anda	vai para
0	0		D	0
0	1		E	13
1	0	1	D	65
1	1	0	D	1
2	0	1	PARE	
2	1		E	66

Note que quando a máquina escreve o que acabou de ler, essa operação pode ser ignorada, já que a fita é única (é de entrada e de saída ao mesmo tempo). Por essa razão nestes casos a coluna "escreve?" fica em branco.

Numeração binária Em lugar de usar 0, 1, 2... para rotular os estados internos, seria melhor usar símbolos formados por 0 e 1s. Usaremos para tanto, o sistema de numeração binário: $0 \rightarrow 0$,
$1 \rightarrow 1$,
$2 \rightarrow 10$,
$3 \rightarrow 11$,
$4 \rightarrow 100$, etc etc

Com esta notação, teríamos agora a seguinte máquina de Turing

Estado atual	Sinal lido	faz o que	Vai para estado	Escreve	Anda para onde
0	0	→	0	0	D
0	1	→	1101	1	E
1	0	→	100000	1	D
1	1	→	1	0	D
10	0	→	0	1	D.PARE
10	1	→	1000010	1	E

Representando números na fita Como pretendemos que os números possam ser fornecidos como input, precisamos de um sistema para escreve-los na fita de entrada. Começaremos escrevendo os números em um sistema unário (1 é 1, 11 é 2, 111 é 3, 1111 é 4, e assim por diante). O símbolo 0 passa a ser o separador.

Soma 1 em unário Vamos escrever uma máquina de Turing para somar 1 em um número unário. Ficaria:

Estado atual	Sinal lido	faz o que	Vai para estado	Escreve	Anda para onde
0	0	→	0	0	D
0	1	→	1	1	D
1	0	→	0	1	Pare
1	1	→	1	1	D

Ou na sua versão mais operacional

estado	chega	escreve?	anda	vai para
0	0		D	0
0	1		D	1
1	0	1	PARE	
1	1		D	1

Devemos notar que toda MT tem a primeira instrução igual, e esta tem a finalidade de fazer avançar a fita pelos zeros não significativos colocados no começo. Apenas quando chega o primeiro 1 é que as coisas começam a acontecer.

Multiplica por 2 em unário Seja a máquina de Turing para multiplicar um número unário por 2.

Estado atual	Sinal lido	faz o que	Vai para estado	Escreve	Anda para onde
0	0	→	0	0	D
0	1	→	1	0	D
1	0	→	10	1	E
1	1	→	1	1	D
10	0	→	11	0	D
10	1	→	100	0	D
11	0	→	0	1	pare
11	1	→	11	1	D
100	0	→	101	1	E
100	1	→	100	1	D
101	0	→	10	1	E
101	1	→	101	1	E

Ou sua versão mais amigável

estado	chega	escreve?	anda	vai para
0	0		D	0
0	1	0	D	1
1	0	1	E	2
1	1		D	1
2	0		D	3
2	1	0	D	4
3	0	1	PARE	
3	1		D	3
4	0	1	E	5
4	1		D	4
5	0	1	E	2
5	1		E	5

mdc de Euclides (em unário) Finalmente, seja uma Maquina de Turing para calcular o algoritmo mdc de Euclides

 ©88-09, Pedro Kantek

Estado atual	Sinal lido	faz o que	Vai para estado	Escreve	Anda para onde
0	0	→	0	0	d
0	1	→	1	1	e
1	0	→	10	1	d
1	1	→	1	1	e
10	0	→	1010	0	d
10	1	→	11	0	d
11	0	→	100	0	d
11	1	→	11	1	d
100	0	→	100	0	d
100	1	→	101	0	d
101	0	→	111	0	e
101	1	→	110	1	e
110	0	→	110	0	e
110	1	→	1	1	e
111	0	→	111	0	e
111	1	→	1000	1	e
1000	0	→	1001	0	e
1000	1	→	1000	1	e
1001	0	→	10	0	d
1001	1	→	1	1	e
1010	0	→	0	0	pare
1010	1	→	1010	1	d

ou no modelo simplificado

estado	chega	escreve?	anda	vai para
0	0		D	0
0	1		E	1
1	0	1	D	2
1	1		E	1
2	0		D	10
2	1	0	D	3
3	0		D	4
3	1		D	3
4	0		D	4
4	1	0	D	5
5	0		E	7
5	1		E	6
6	0		E	6
6	1		E	1
7	0		E	7
7	1		E	8
8	0		E	9
8	1		E	8
9	0		D	2
9	1		E	1
10	0		PARE	
10	1		D	10

Sistema contraído O sistema unário é ineficiente, logo precisamos usar um sistema binário. Só que não podemos faze-lo diretamente, já que se assim for, não saberemos quais as marcas de fim e quais os zeros para valer. Para resolver esta questão, vamos fazer uma contração. Qualquer grupo de zeros e uns não é lido diretamente como binário, e sim cada quantidade finita de uns é substituida por 2, 3 ... na segunda seqüência. Ou seja na segunda seqüência, cada número representa o número de UNS entre ZEROS

Antes de prosseguir, vamos entender bem esta passagem. Seja a sequencia

```
010.000101101010110100011101010111100110   = 1
010.0.001011010101101000111010101111100110  = 0
0100.0.010110101011010001110101011110011O  = 0
01000.010.110101011010001110101011110011O  = 1
0100001.0110.101011010001110101011110011O  = 2
0100001011.010.10110100011101010111100110  = 1
010000101101.010.11010001110101011110011O  = 1
01000010110101.0110.1000111010101111100110  = 2
0100001011010101l.010.001110101011110011O  = 1
010000101101010110l0.0.01110101011110011O  = 0
01000010110101011010.0.0111010101111001l0  = 0
0100001011010101l0100.0.111010101111001l0  = 0
0100001011010101l0100.0111.01010111100110  = 3
0100001011010101101000111.010.10111100110  = 1
010000101101010110100011101.010.111100110  = 1
0100001011010101101000111010l.011110.0110  = 4
010000101101010110100011101010111l0.0.110  = 0
01000010110101011010001110101011110.0110.  = 2
```

Note a diferença:

```
01110111 dá 33;  011100111 dá 303
```

A seqüência acima ficou sendo:

```
1 0 0 1 2 1 1 2 1 0 0 3 1 1 4 0 2
```

Note-se que agora podemos designar aos números códigos de acordo com o que quisermos fazer. Assim, por exemplo, o número 2 pode passar a ser uma vírgula.,. o 3 pode ser menos, o 4 mais, o 5 vezes, ...

A linha original seria lida como:

9 , 3 , 4 instrução-3 3 instrução-4 0 ,

Para quem está perdido, acompanhe: 1001 (09), 2 (virgula), 11 (3), 2 (vírgula), 100 (quatro), 3 (Instrução3), 11 (3), 4 (instrução4), 0 (0), 2(vírgula).

Este procedimento nos permite terminar um número, colocando uma vírgula ao final.

Vamos fazer o procedimento contrário, o da expansão:

Seja escrever a sequencia de números inteiros: 5, 13, 0, 1, 1, 4. Em notação binária, tais números ficam: 101, 1101, 0, 1, 1, 100, que expandido na fita, dará:

```
...0000...1001011010100101100110101101011010001l0..000...
```

Para realizar esta codificação diretamente, usemos a equivalencia:
$0 \rightarrow 0$
$1 \rightarrow 10$
, $\rightarrow 110$
e acrescentando zeros à vontade à esquerda e à direita, dica

```
...000...10 0 10 110 10 10 0 10 110 0 110 10 110 10 110 10 0 0 110...0...
 zeros   1  0 1  ,   1  1  0 1  ,   0 ,   1  ,   1  ,   1  0 0 ,   zeros
```

 ©88-09, Pedro Kantek

Chamamos a esta última notação de notação binária desdobrada.

Uma observação final, é de que os zeros à esquerda são dispensáveis. Assim, e por via de conseqüência, também o zero isolado é dispensável, podendo-se escrever apenas „ significando estas duas vírgulas que há aí um zero entre elas.

Codificando isto da maneira como foi vista, a seqüência ficaria 101,1101,,1,1,100 e desdobrada ficaria

```
...0000...10010110101001011011010110101101000110...000...
```

note-se que sumiu um zero no meio (o local correto está marcado com um _).

```
...0000...10010110101001011_1101011010110100011O...000...
```

Examinemos agora uma Máquina de Turing para rodar o algoritmo de euclides: Por exemplo, para o par 6,8 que vimos antes teríamos que usar

```
...00000...111111011111111...0000000...
```

Mas usando as representações vistas acima, o par ficaria 6,8 em binário 110,1000, que por expansão é codificado como

...000...10 10 0 110 10 0 0 0 110...000...

Para este par específico não há muito ganho, mas supondo que os números a representar fossem: 1583169,8610 em notação binária dariam 110000010100001000001,10000110100010, e na fita iriam como ...000...10100000010010000010000001011010000010100100000100110...000... ou ...000... 10 10 0 0 0 0 0 10 0 10 0 0 0 0 10 0 0 0 0 0 10 110 10 0 0 0 0 10 10 0 10 0 0 0 10 0 110 ...000... ou

Na notação unária estes 2 números precisariam 1.6 Mcaracteres (umas 800 páginas de livro).

Adiciona 1 em notação expandida Naturalmente, a MT poderia pegar estes números contraídos e desdobrá-los em representação unária antes de operar. Mas nada impede que façamos uma MT que já trabalhe direto com números expandidos. Fazê-la para o algoritmo de Euclides seria muito complicado, então vamos a um exemplo mais simples. Seja a máquina que adiciona 1 ao número dado (em notação expandida)

Estado atual	Sinal lido	faz o que	Vai para estado	Escreve	Anda para onde
0	0	→	0	0	d
0	1	→	1	1	d
1	0	→	0	0	d
1	1	→	10	1	d
10	0	→	11	0	e
10	1	→	10	1	d
11	0	→	0	1	pare
11	1	→	100	0	e
100	0	→	101	1	e
100	1	→	101	1	e
101	0	→	110	0	d
101	1	→	10	1	d
110	0	→	...	...	...
110	1	→	111	1	d
111	0	→	11	1	d
111	1	→	111	0	d

... esta linha não vai ocorrer nunca nesta MT razão pela qual é irrelevante o que se coloca nela. Mais tarde, quando tivermos que numerar as MTs, será necessário preencher este espaço. Pode-se colocar qualquer coisa aí.

Para testar, use o número 167, cuja representação binária é 10100111 e expandido é ...0000... 10 0 10 0 0 10 10 10 110 ...0000... Supondo 167+1=168, fica (em binário normal) 10100111 + 1 = 10101000. 168 expandido é: ...0000... 10 0 10 0 10 0 0 0 110 ...00000...

Use esta configuração de trabalho

estado	chega	escreve?	anda	vai para
0	0		D	0
0	1		D	1
1	0		D	0
1	1		D	2
2	0		E	3
2	1		D	2
3	0	1	PARE	
3	1	0	E	4
4	0	1	E	5
4	1		E	5
5	0		D	6
5	1		D	2
6	0	?	?	?
6	1		D	7
7	0	1	D	3
7	1	0	D	7

Multiplica por 2 em notação expandida Uma maquininha simples, onde paradoxalmente trabalhar com números expandidos é mais fácil do que com números unários é a máquina que multiplica um número por 2.

Estado atual	Sinal lido	faz o que	Vai para estado	Escreve	Anda para onde
0	0	→	0	0	d
0	1	→	1	1	d
1	0	→	0	0	d
1	1	→	10	0	d
10	0	→	11	1	d
11	0	→	0	0	pare

Ou a versão de trabalho

estado	chega	escreve?	anda	vai para
0	0		D	0
0	1		D	1
1	0		D	0
1	1	0	D	2
2	0	1	D	3
3	0		PARE	

Tese de Church-Turing Tendo estudado as máquinas de Turing podemos afirmar serem elas capazes de realizar qualquer computação.

Turing, gastou um bocado de tutano provando isso. Veio em seu apoio a idéia de Alonso Church, lógico americano, que com a ajuda de Kleene apresentou um esquema para resolver o 10. problema de Hilber chamado λ-cálculo. Outra proposta foi a de Emil Post, lógico polonês-americano. Viu-se logo que todos estes esquemas eram equivalentes. Isto deu origem à tese de Church-Turing, de que a máquina de Turing define o que

 ©88-09, Pedro Kantek

matematicamente entendemos por um procedimento algoritmico. (A tese diz que "as funções recursivas enumeráveis são únicas".)

Traduzindo para o português comum, a tese diz que para qualquer problema algorítmico há uma máquina de Turing que o resolve e que cada Máquina de Turing resolve um problema algorítmico.

Máquina Universal de Turing O princípio é simples. Teremos uma máquina U, que antes de mais nada lerá as informações na fita de entrada para que ela passe a se comportar como a máquina T. Depois disso vem os dados de input que seriam processados pela máquina T, mas que serão pela U, operando como se fosse a T.

Para ver isso funcionando, precisamos numerar as máquinas de Turing. É preciso um pouco de esperteza aqui. Primeiro, convertamos D, E, PARE, → e vírgula como os números 2, 3, 4, 5 e 6. Ou nas nossas contrações como 110, 1110, 11110, 111110 e 1111110. Zero será 0, e 1 será 10, como já vimos. As colunas 4 e 5 nas nossas tabelas não precisam separação, já que o conteúdo da coluna 5 é apenas 1 ou zero e sempre ocupa 1 caracter.

Podemos nos dar ao luxo de não codificar nenhuma seta, nem nada do que as antecede (colunas 1, 2 e 3) desde que organizemos as ordens em rigorosa ordem crescente (e tomando cuidado para preencher ordens que no modo tradicional não eram escritas por nunca ocorrerem na máquina).

Fazendo tudo isso, e escrevendo toda a programação da máquina que soma 1 a um número unário, ela fica:

```
1010110110100101101010011101001011010111101000011101001010110101000101110101000110100101101101010101101010101101010100.
```

Convertendo este número binário para decimal chegamos a 450813704461563958982113775643437908
O que significa que a máquina que soma 1 em unário é a 450. 813. 704. 461. 563. 958. 982. 113. 775. 643. 437.908^a máquina de Turing. Naturalmente, mudando-se (otimizando-se) alguma coisa na programação, esta máquina muda de número.

A máquina que soma 1 em unário é a 177.642^a máquina. A que multiplica 2 em binário expandido é 10.389.728.107^a máquina e a que multiplica por 2 um número unário é a 1. 492. 923. 420. 919. 872. 026. 917. 547. 669 ^{a} máquina.

Dentro desta perspectiva qual seria a máquina T_0 ?

Estado atual	Sinal lido	faz o que	Vai para estado	Escreve	Anda para onde
0	0	→	0	0	d
0	1	→	0	0	d

movimenta-se para a direita, apagando tudo o que encontra, sem nunca parar e sem nunca voltar.

E T_1 ?

Estado atual	Sinal lido	faz o que	Vai para estado	Escreve	Anda para onde
0	0	→	0	0	d
0	1	→	0	0	e

Faz a mesma coisa que T_0, só que pula 1 para trás depois de apagar cada marca da fita.

E T_2 ?

Estado atual	Sinal lido	faz o que	Vai para estado	Escreve	Anda para onde
0	0	→	0	0	d
0	1	→	0	1	d

Se movimenta interminavelmente para a direita, mas deixa tudo como está Nem a 0, nem a 1 nem a 2 são máquinas de Turing, já que não param nunca.

T_3

Estado atual	Sinal lido	faz o que	Vai para estado	Escreve	Anda para onde
0	0	→	0	0	d
0	1	→	0	0	pare

Esta é a primeira máquina. Ela para depois de mudar o 1 mais a esquerda em zero.

E assim vai. A T_7, é não corretamente especificada, já que existem uma seqüência de 5 uns, e ela emperra ao processar tal seqüência.

Dizendo que quando a n-ésima máquina de Turing atua sobre o número m produz o número p, podemos escrever $T_n(m) = p$. Podemos pensar nesta relação como sendo uma função de 2 parâmetros (n e m) que leva a p. Esta função, pelo que vimos é totalmente algoritmica. Logo, ela pode ser realizada por uma máquina de Turing, a quem chamaremos de U. Então, fornecendo o par (n, m) a U, obtemos como resposta p. Vamos escrever n e m em uma fita, separando-os pela seqüência (nova) 111110.

Por exemplo, para n=11 e m=6, a fita ficaria

```
...000...1011 111110 110 ...1000...
```

Podemos escrever $U(n, m) = T_n(m)$

A máquina U quando alimentada primeiro com o número n, passa a se comportar como a T_n

Como U é uma máquina de Turing, ela é uma $T - alguma - coisa$. Quanto é essa alguma coisa ?

É mais ou menos 7.24×10^{1688} , ou seja o número 724 seguido de 1686 zeros.

Todos os computadores modernos, desde um humilde Z80 até um Cray multi-vetorial são máquinas de Turing.

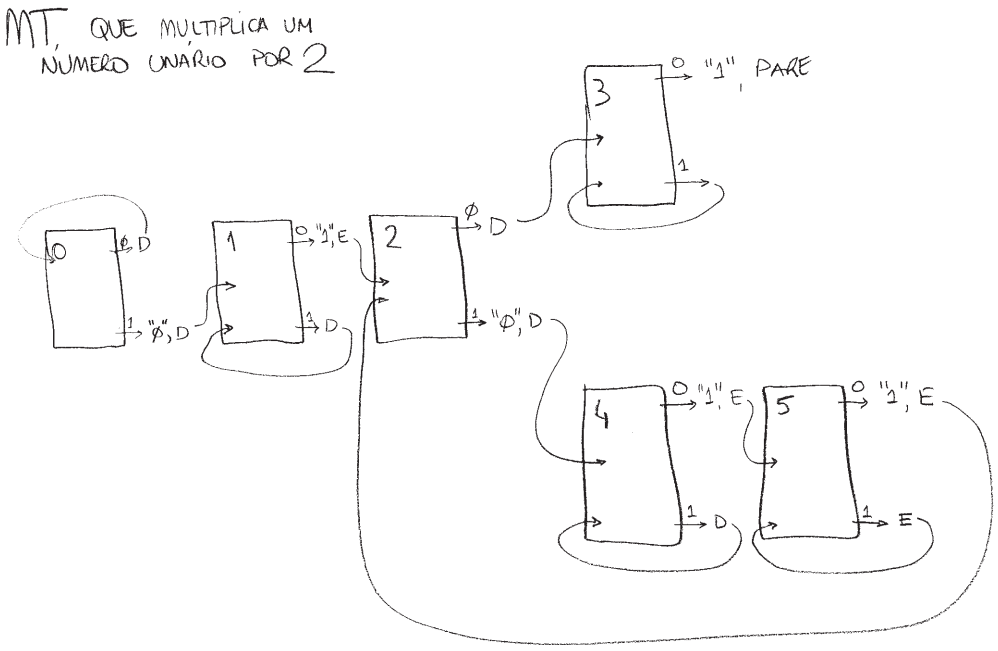
Não sei se é óbvio perceber, mas esta idéia é uma das mais maravilhosas idéias que uma mente humana (humana ? há controvérsias: há quem diga que Turing era um ET meio de passagem por aqui.... ;-) produziu: acabamos de ser apresentados a um moderno computador: em $U(n, m) = p$, U é o hardware, n é o software (ou programa), m são os dados de entrada e finalmente p é o resultado esperado.

Representação gráfica da Máquina de Turing

Para efeito de entender a MT, pode-se desenhar algo como segue: Usar como modelo a MT que multiplica por 2 em unário Construir o modelo acima no quadro e depois testar as sequencias 000010000, 00001100000 e 00001110000. Todas vão funcionar.

No WS VIVO802 tem uma função chamada “simula” que recebe uma MT na forma de uma matriz nn × 5, no formato

 ©88-09, Pedro Kantek



estado	chega	escreve?	anda	vai para
0	0		D	0
0	1		E	1
1	0	1	D	2
1	1		E	1
2	0		D	10
2	1	0	D	3
3	0		D	4
3	1		D	3
4	0		D	4
4	1	0	D	5
5	0		E	7
5	1		E	6
6	0		E	6
6	1		E	1
7	0		E	7
7	1		E	8
8	0		E	9
8	1		E	8
9	0		D	2
9	1		E	1
10	0		PARE	
10	1		D	10

Esta matriz já está no WS com o nome de MDCUN. Forma de chamar MDCUN simula
'00000011110111111000000000000' e a resposta será '0000011000000000'.

Outro tema diretamente ligado à Teoria de Turing é

2.2 Teorema de Goedel

Figura 2.1: Kurt Göedel

Gödel nasceu em 18 de abril de 1906 em Brno atual Rep. Tcheca. Durante toda a sua vida escolar somente uma única vez Gödel deixou de ganhar a nota máxima (e logo em matemática). Foi batizado de criança como *der Herr Warum* (O senhor Por que ?). Apresentou sua tese de doutorado em Viena em 1929. Morreu em 14/01/78 de fome.

Em fins do século XIX, no Congresso de Matemática de Paris, em 1900, Hilbert, renomado professor em Göttingen apresentou 23 problemas, que segundo ele estariam ocupando os matemáticos do século XX. Seu segundo problema perguntava se é possível provar que os axiomas da aritmética são consistentes, isto é, dado um número finito de passos lógicos corretos, nunca se chegará a uma contradição. Na esteira desse problema, Bertran Russell, um filósofo matemático (ou será matemático filósofo ?) em 1910, começou uma série de livros (Principia Mathematica), na qual, baseado nos axiomas de Peano, desenvolvia todo um programa de formalização da matemática. A tentação era saborosa: provar que lógica e matemática eram a mesma coisa. Mas, o segundo problema de Hilbert continuava em aberto. Estudos posteriores de um jovem matemático austríaco, Kurt Gödel, emigrado para os EUA, que se consubstanciaram no teorema que leva o seu nome, deram o tiro de misericórdia na pretensão. Gödel provou que num sistema lógico formal existem assertivas verdadeiras que não podem ser provadas. Foi mais um sopetão na pretensão homocêntrica do Universo (assim como fizeram Copérnico, Darwin e Freud, só para ficar nos mais evidentes). Antes de passar ao Teorema de Gödel, diga-se apenas que dos 23 problemas, aproximadamente a metade ainda não tem solução (terá algum dia ?), e pior, ou melhor, a matemática se desenvolveu em centenas de direções nem sonhadas por Hilbert. Ainda bem que o trabalho final dele, terminava com a citação: *Enquanto uma ciência oferece abundância de problemas, ela está viva.*

Aliás, buscando inspiração no Congresso de 1900, na virada do século XX para o XXI, um americano rico, deixou 1 milhão de dólares para quem resolver por primeiro 7 problemas ainda pendentes na matemática: O problema P versus NP, além da hipótese de Riemann, a teoria de Yang-Mills (hipótese de lacuna de massa), as equações de Navier-Strokes, a conjectura de Poincaré, a conjectura de Birch, Swinnerton e Dyer e a conjectura Hodge) [2]

[2] vide em www.claymath.org as regras do prêmio e a descrição formal deste e de mais 6 problemas:

 ©88-09, Pedro Kantek

Demonstração do Teorema de Incompletude

Seu teorema se baseia em um processo de 3 etapas:

1. Estabelecer as regras pelas quais axiomas (e teoremas) podem gerar novos teoremas
2. Estabelecer os axiomas da aritmética em linguagem de lógica de predicados
3. Estabelecer uma forma de numeração para os teoremas assim gerados.

Eis os axiomas de Peano para os números naturais. Note-se que o símbolo s denota sucessor, um conceito primitivo. O sucessor de 0 é 1, de 1 é 2, ... de n é n+1:

$\forall x \sim (0 = sx)$	Não existe x tal que 0 seja seu sucessor
$\forall x, y(sx = sy) \Rightarrow (x = y)$	Se 2 números tem o mesmo sucessor, são iguais
$\forall x, x + 0 = x$	0 é o elemento neutro na adição
$\forall xy, x + sy = s(x + y)$	x somado com o sucessor de y é igual ao sucessor de x+y
$\forall x, yx \times sy = xy + x$	x=2, y=5 leva a $2 \times 6 = 2 \times 5 + 2 = 12$
$\forall x, x \times 0 = 0$	0 é o elemento absorvente na multiplicação
$\forall x, x = x$	identidade
$\forall xyz, (x = y) \Rightarrow ((x = z) \Rightarrow (y = z))$	propriedade transitiva
$\forall xy, (x = y) \Rightarrow (A(x, x) \Rightarrow A(x, y))$	A é qualquer fórmula com 2 variáveis livres
$(P(0) \wedge \forall x P(x) \Rightarrow P(sx))) \Rightarrow \forall x P(x)$	Regra fundamental da indução matemática

Com estas informações, Gödel, passou à etapa 3 do teorema, usando a tabela a seguir:

Símbolo	Código	Símbolo	Código	Símbolo	Código
0	1	(	6	$\sim$	11
s	2	)	7	$\wedge$	12
+	3	,	8	$\exists$	13
$\times$	4	x	9	$\forall$	14
=	5	1	10	$\Rightarrow$	15

Note-se que só existe a variável x. Logo quando só aparece x, ele será identificado como x_1. Quando aparecerem x e y, eles serão x_1 e x_{11}, e assim por diante.

Agora para cada cláusula, Gödel bolou um número, que mais tarde foi chamado "Número de Gödel", e que é construído da seguinte maneira. Tomam-se os primos a partir do 2. São eles: 2, 3, 5, 7, 11, 13, 17, 23, 29... Cada axioma terá seu número de Gödel calculado por um sistema de numeração onde as bases são os primos e os expoentes são o numero da tabela acima. Para clarear, vejamos o número de Gödel do axioma 4 de Peano:

$\forall x, y; x + sy = s(x + y)$ [Escrito em linguagem matemática]

$x_1 + sx_{11} = s(x_1 + x_{11})$ [Escrito na forma adequada para o teorema de Gödel],

e o número é:

$2^9.3^{10}.5^3.7^2.11^9.13^{10}.17^{10}.19^5.23^2.29^6.31^9.37^{10}.41^3.43^9.47^{10}.53^{10}.59^7$

Note-se que o numero é enorme, mas isso é de propósito. É para que, dado um número, SE ELE FOR UM NÚMERO DE GÖDEL, possa-se recuperar qual o axioma ou teorema que lhe deu origem. Basta fatorá-lo em seus componentes primos, e verificar qual o expoente de cada um dos primos. Ou seja, a relação axioma - número de Gödel é bi-unívoca.

Estamos chegando perto da prova:

Seja o predicado PROVA(x,y,z), lido como x é o número de Gödel da prova da fórmula Y (que tem número de Gödel y), o qual teve o inteiro z inserido dentro dela. Note-se que PROVA(x,y,z) é um jeito fácil de escrever uma imensa e longa expressão onde aparecem x1, x11 e x111,

Essa expressão inclui muitos procedimentos, entre eles:

- dado um inteiro, fatore-o em seus fatores primos
- ache a expressão que lhe deu origem
- verifique se a expressão verifique se ela é uma fórmula
- verifique se ela esta provada

Claramente são procedimentos irrealizáveis na prática, mas em princípio, computáveis.

Vamos considerar um caso especial do predicado acima: Suponha que a fórmula Y é alimentada com seu próprio número de Gödel, e vamos tentar negar a existência de tal prova. Escreve-se $\sim \exists\, xPROVA(x, y, y)$

Em palavras: x é o número de Gödel da prova obtida na fórmula Y pelo número y (Número de Gödel de Y). Vamos chamar ao número de Gödel da PROVA(x,y,y) de g.

Finalmente: o TEOREMA DE GÖDEL

$\sim \exists\, xPROVA(x, g, g)$ é verdadeiro, mas não é provável (no sentido de poder ser provado) no sistema aritmético formal.

A demonstração ocupa poucas linhas:

Suponha que $\sim \exists\, xPROVA(x, g, g)$ é provável (no mesmo sentido) e seja p o número de Gödel dessa prova P. Então, nós temos que PROVA(p,g,g) é verdadeiro, desde que P é a prova de G, na qual foram substituídas as variáveis livres. MAS, a veracidade de PROVA(p,g,g) contradiz $\sim \exists\, xPROVA(x, g, g)$, e daqui temos que não existe essa prova.

Então, se tudo foi construído corretamente, a afirmação $\sim \exists\, xPROVA(x, g, g)$ é verdadeira e portanto não existe a $PROVA(x, g, g)$ dentro do sistema.

Em resumo: Dentro de um sistema de lógica formal existem teoremas que são indecidíveis.

O teorema de Goedel decide as duas primeiras questões de Hilbert: a matemática não é completa e e nem pode ser provada consistente, mas não resolve a terceira questão. Esta foi respondida negativamente de forma independente por dois cientistas: o primeiro denominado Alonso Churchs em abril de 1936 na universidade de Princeton ano nos Estados Unidos. Outro foi Alan Mathison Turing, em agosto de 1936 no Kings College em Londres. Um terceiro matemático, o polonês naturalizado americano Emil Post, professor do City College de New York submeteu em outubro de 1936 um artigo ao Jornal de Lógica Simbólica no qual expõe uma idéia similar à de Turing, embora menos ambiciosa.

Do trabalho de Church derivou-se a notação λ para definir funções, que está por trás da construção do interpretador LISP. O trabalho do Post deu origem ao PROLOG. O trabalho de Turing é simplesmente o projeto de um moderno computador digital.

2.2.1 Desafio

- Coloque no papel qual seria um programa baseado em IA que traria riquezas e fama (à la Windows/Bill Gates) ao seu criador.
- Faça uma avaliação histórica do trabalho de Allan Turing sobre as Máquinas de Turing vis-a-vis a moderna informática.

 ©88-09, Pedro Kantek

Pode uma máquina pensar?

Por Alan M Turing, 1950

Tradução e adaptação dos seis primeiros parágrafos do texto "Can a Machine Think"de A. M. Turing, in J. R. Newman (ed.) The World of Mathematics - A Small Library of the Literature of Mathematics from A'h-mosé the Scribe to Albert Einstein Vol. 4, pp. 2075-2092. Tradução de Rute Queiroz Mesquita. Revisão de Olga Pombo. Adaptação de Pedro Kantek.

1. O jogo da imitação
2. Crítica do novo problema
3. As máquinas que interessam no jogo
4. Computadores digitais
5. A Universalidade dos computadores digitais
6. Visões Contrárias à questão fundamental
   (a) A objeção teológica
   (b) Objeção das cabeças na areia
   (c) Objeção matemática
   (d) O argumento da consciência
   (e) Argumentos a partir de várias incapacidades
   (f) Objeção de Lady Lovelace.
   (g) Argumento sobre a continuidade no sistema nervoso
   (h) O argumento da informalidade do Comportamento
   (i) Argumento da Percepção Extra-Sensorial

O jogo da imitação

Proponho que consideremos a questão: "Podem as máquinas pensar?". Deveríamos começar com as definições do significado dos termos: "máquina"e "pensar". Estas definições poderiam ser elaboradas de forma a refletir, o melhor possível, o uso normal das palavras. Contudo, esta posição é perigosa.

Se quiséssemos encontrar o significado das palavras "máquina"e "pensar"analisando o modo como estas são normalmente utilizadas, seria difícil escapar à conclusão de que o significado e a resposta para a questão "Podem as máquinas pensar?"deveria ser visto como um estudo estatístico, tal como quando se ausculta a opinião pública. O que é um absurdo. Em vez de procurar tal definição, irei substituir a questão por outra, intimamente ligada com a primeira e expressa em palavras relativamente claras.

A nova forma do problema pode ser descrita nos termos de um jogo, a que chamaremos "jogo de imitação". Joga-se com três pessoas: um homem (A), uma mulher (B) e um interrogador (C) que pode ser de ambos os sexos. O interrogador fica numa sala à parte dos outros dois. O objetivo do jogo para o interrogador é determinar qual dos outros dois participantes é o homem e qual é a mulher. Ele identifica-os pelas etiquetas X e Y e, no final do jogo, diz se "X é A e Y é B"ou "X é B e Y é A".

O interrogador pode colocar questões a A e B da seguinte forma:

C: Pode X por favor dizer-me qual o tamanho do seu cabelo?

Suponhamos agora que X é realmente A. Então A tem de responder. O objetivo do jogo para A é tentar fazer com que C faça uma identificação errada. A sua resposta poderia por isso ser: "O meu cabelo é escadeado, mas os fios mais longos têm aproximadamente 20cm".

Para que o tom da voz não ajude o interrogador, as respostas devem ser escritas, ou melhor ainda, tipografadas. A solução ideal é ter uma teleimpressora que comunique entre as duas salas. Como alternativa, as perguntas e respostas podem ser repetidas por um intermediário.

O objetivo do jogo para o terceiro jogador (B) é ajudar o interrogador. Provavelmente, a melhor estratégia para ela é dar respostas verdadeiras. Ela pode acrescentar às suas respostas, coisas do tipo: "Eu sou a mulher, não lhe dê ouvidos!". Mas isso não lhe trará qualquer proveito, visto que o homem poderá fazer observações semelhantes.

Perguntamos então: "O que acontecerá quando uma máquina toma o lugar de A no jogo?". Será que, quando o jogo se desenrola desta maneira, o interrogador vai decidir incorretamente tantas vezes quantas como quando o jogo é jogado entre um homem e uma mulher?

São estas as perguntas que substituem a nossa questão original: "Podem as máquinas pensar?".

Crítica do novo problema

Da mesma forma que perguntamos: "Qual é a resposta para esta nova forma da questão?", poderíamos perguntar: "Será que esta é uma questão que vale a pena investigar?". Investigaremos esta última questão sem mais cerimonias de forma a pôr fim a uma regressão ao infinito.

O novo problema tem a vantagem de desenhar uma linha bem definida entre as capacidades físicas e intelectuais de um homem.

Nenhum engenheiro ou químico reivindica o fato de ser capaz de produzir um material que seja indistingüível da pele humana. É possível que um dia isto possa vir a ser realizado. Contudo, mesmo supondo que esta invenção estivesse disponível, sentiríamos que tinha pouco sentido tentar fazer uma "máquina que pensa"mais humana, vestindo-a com um corpo artificial.

A forma como colocamos o problema reflete este fato na condição uma vez que impede o interrogador de ver ou tocar os outros participantes, ou ainda de ouvir as suas vozes.

Outras vantagens do critério proposto podem ser reveladas através de questões e respostas exemplares. Assim:

> Q: Por favor, escreva-me um soneto sobre o tema "Ponte levadiça"
>
> R: Não considere esta pergunta. Nunca consegui escrever poesia.
>
> Q: Some 34957 a 70764.
>
> R: (Faz uma pausa de cerca de 30 segundos e então dá a resposta) 105621.
>
> Q: Sabe jogar xadrez?
>
> R: Sim.
>
> Q: Eu jogo com as pretas. Tenho o rei em g1 e mais nenhuma peça. Você tem apenas o rei em g3 e uma torre em a8. É a sua vez de jogar. Qual vai ser a sua jogada?
>
> R: (Depois de uma pausa de 15 segundos) torre em a1++. Xeque-mate.

O método de pergunta e resposta parece ser apropriado para introduzir quase todas as áreas da atividade humana que queiramos incluir.

 ©88-09, Pedro Kantek

Não queremos penalizar a máquina pela sua incapacidade de brilhar em competições de beleza, nem penalizar o homem por perder numa corrida contra um avião. As condições do nosso jogo tornam estas incapacidades irrelevantes. As "testemunhas"podem gabar-se, tanto quanto desejarem, sobre o seu encanto, força ou heroísmo, mas o interrogador não pode exigir demonstrações práticas.

Talvez o jogo possa ser criticado pelo fato de as probabilidades de sucesso pesarem demasiado contra a máquina. Se o homem fosse posto à prova e pretendesse ser a máquina, certamente faria uma demonstração muito pobre. Denunciar-se-ia imediatamente pela lentidão e inexatidão nos cálculos matemáticos. Será que as máquinas não poderão levar a cabo algo que deva ser descrito como pensar, mas que é muito diferente do que o homem faz?

Esta objeção é bastante forte. Contudo, podemos dizer que se uma máquina puder ser construída para jogar, de forma satisfatória, o "jogo da imitação", então não necessitamos nos preocupar com esta objeção. Pode alegar-se que quando se joga o "jogo da imitação", a melhor estratégia para a máquina seja não imitar o comportamento humano. Pode ser verdade, mas penso ser improvável haver qualquer efeito deste tipo. Seja como for, não temos intenção de investigar a teoria do jogo e assumimos que a melhor estratégia é tentar arranjar respostas que seriam naturalmente dadas por um homem.

As máquinas que interessam no jogo

A questão que colocamos no parágrafo 1 não estará completamente definida até que especifiquemos o que queremos dizer com a palavra "máquina". É natural que queiramos permitir que qualquer tipo de técnica de engenharia possa ser utilizada nas nossas máquinas. Também desejamos dar a possibilidade de que, um engenheiro ou uma equipe de engenheiros, possam construir uma máquina que funcione, mas cuja forma de operar não possa ser satisfatoriamente descrita pelos seus construtores, visto terem aplicado um método amplamente experimental. Por último, desejamos excluir das máquinas, homens que nasceram da forma tradicional. É difícil modelar as definições de forma a satisfazerem estas três condições. Podia, por exemplo, insistir-se em que a equipe de engenheiros fosse toda do mesmo sexo. O que não seria satisfatório, visto que existe a probabilidade de se poder criar um indivíduo completo, a partir de uma única célula da pele de um ser humano. Realizar isto, seria um feito da técnica biológica, digno do mais alto louvor, mas não estamos inclinados a reconhecer isto como um caso de "construção de uma máquina que pensa". O que nos leva a abandonar a exigência de que todo o tipo de técnica deveria ser permitido. Somos os mais preparados para fazer isto devido ao presente interesse pelas "máquinas que pensam" ter surgido a partir de um tipo de máquina particular, habitualmente chamado de "computador electrônico" ou "computador digital". Seguindo esta sugestão, apenas permitiremos que os computadores digitais tomem parte no nosso jogo.

Esta restrição pode parecer à primeira vista, muito drástica. Procurarei demonstrar que, na realidade, não é tanto assim. Mas para fazer isso, é necessário uma pequena consideração relativa à natureza e propriedades destes computadores.

Pode também dizer-se que esta identificação das máquinas com os computadores digitais, como no nosso critério de "pensar", só não será satisfeito se (contrariamente ao que creio) os computadores digitais não forem capazes de fazer uma boa exibição no jogo.

Já existem alguns computadores digitais prontos para funcionar e pode então fazer-se a pergunta: "Porque não tentamos experimentá-los já?". Seria fácil satisfazer as condições do jogo. Podia usar-se um determinado número de interrogadores e compilarem-se as estatísticas, para mostrar quantas vezes foi feita a identificação correta. A resposta breve seria que não estamos perguntando se todos os computadores digitais procederiam

bem no jogo, nem se os computadores atualmente disponíveis também procederiam bem, mas se há computadores imaginários capazes de o fazer. Estamos apenas perante uma resposta breve. Posteriormente, analisaremos esta questão sob outro ponto de vista.

Computadores digitais

A ideia por detrás dos computadores digitais pode ser explicada dizendo que estas máquinas pretendem levar a cabo qualquer operação que possa ser feita por um computador humano. É suposto que o computador humano siga um conjunto de regras fixas. Ele não tem autoridade para desviar de si nenhum detalhe. Podemos supor que estas regras são fornecidas num livro, o qual é alterado sempre que surgir uma nova tarefa. Também tem um abastecimento de papel ilimitado, no qual faz os seus cálculos. Poderá também fazer as multiplicações e adições num "computador de secretária", mas isto não é importante.

Se utilizarmos a explicação acima como uma definição, correremos o risco de andar à volta na argumentação. Evitamos isto dando um esboço das formas pelas quais é conseguido o efeito desejado. Um computador digital pode ser normalmente visto como sendo constituído por três partes.

I - Armazenamento

II - Unidade de processamento

III - Unidade de Controle.

O armazenamento é um depósito de informação e corresponde ao papel que é utilizado por um computador humano, quer para fazer os seus cálculos, quer para imprimir o seu livro de regras. À medida que um computador humano faz os seus cálculos na sua cabeça, uma parte do armazenamento corresponde à sua memória.

A unidade de processamento é a parte que leva a cabo as várias operações individuais, envolvidas num cálculo. O que elas são, vai variar de máquina para máquina. Normalmente, as operações razoavelmente compridas podem ser feitas, como por exemplo "multiplicar 3540675445 por 7076345687" mas, nalgumas máquinas, apenas podem ser feitas operações muito simples, como por exemplo "registrar 0".

Mencionamos que o "livro de regras" fornecido ao computador é substituído na máquina por uma parte do armazenamento. É chamado a "tabela de instruções". O dever da unidade de controle é verificar se estas instruções são obedecidas e na ordem correta. A unidade de controle é construída de forma a que isto necessariamente aconteça.

A informação no armazenamento é habitualmente partida em pacotes de um tamanho razoavelmente pequeno. Numa máquina, por exemplo, um pacote pode consistir em dez dígitos decimais. São atribuídos números de uma forma sistemática, às partes do armazenamento nas quais os vários pacotes de informação são guardados. Uma instrução típica pode dizer: "Adicionar um número guardado na posição 6809 ao que está na 4302 e colocar o resultado na última posição de armazenamento". Escusado será dizer, que isto não ocorrerá na máquina expresso em português. É mais provável que seja codificado da seguinte forma: 6809430217. O número 17 indica qual das operações possíveis é para executar entre os dois números. Neste caso, a operação é a descrita acima "Adicionar o número...". Reparemos que a instrução ocupa 10 dígitos, formando assim um pacote de informação, o que é muito conveniente. A unidade de controle vai buscar as instruções que devem ser executadas, em seqüência, segundo a posição pela qual foram guardadas. Mas ocasionalmente, pode ser encontrada uma instrução como por exemplo: "Agora executar a instrução guardada na posição 5606 e continuar a partir dali" ou ainda "Se a posição 4505 contém 0, executar a próxima instrução, guardada em 6707, caso contrário, continuar em frente". Instruções deste tipo são muito importantes porque tornam possível repetir uma seqüência de operações vezes sem conta, até se verificar alguma condição. Considerando um exemplo doméstico, suponhamos que a mãe quer que o Toni vá ao sapateiro todas as manhãs, no seu caminho para a escola,

 ©88-09, Pedro Kantek

para saber se os seus sapatos estão prontos. Ela pode relembrá-lo todas as manhãs. Alternativamente, ela pode colocar um aviso no corredor de forma a que, sempre que ele saia de manhã para a escola, leia o aviso a lembrá-lo de passar no sapateiro e também de destruir o bilhete quando chegar em casa e trouxer os sapatos.

O leitor deve aceitar como verdade que um computador digital poder ser construído e de fato têm sido construídos, de acordo com os princípios que descrevemos e que eles podem realmente imitar de uma forma muito aproximada as ações de um computador humano.

O livro de regras que nós descrevemos como sendo usado pelo nosso computador humano, é claro uma conveniente ficção. Os atuais computadores humanos lembram-se o que têm para fazer. Se alguém quer fazer uma máquina imitar o comportamento de um computador humano, nalgumas operações complexas terá de perguntar-lhe como é feito e então traduzir a resposta num formato de tabela de instruções. Construir tabelas de instruções é normalmente descrito como "programar". "Programar uma máquina para executar a operação A", significa pôr a tabela de instruções apropriada na máquina, de forma a que esta execute a operação A.

Uma variante interessante da idéia de computador digital é um "computador digital com um elemento aleatório". Estes têm instruções que involvem o lançamento de um dado ou outro processo equivalente. Uma dessas instruções pode ser por exemplo "Atirar o dado e pôr o número resultante no armazenamento 1000". Por vezes, uma máquina destas é descrita como tendo livre vontade (embora eu próprio não utilize esta expressão). Ao observar um máquina, normalmente não é possível determinar se ela tem um elemento aleatório, pois um efeito semelhante pode ser produzido por esses dispositivos ao fazerem as escolhas depender dos dígitos decimais de número qualquer.

Atualmente, a maior parte dos computadores digitais tem um armazenamento finito. Não há uma dificuldade teórica na idéia de um computador com uma capacidade de armazenamento infinito. Claro que apenas uma parte finita pode ser usada num dado momento. De igual modo, apenas uma pequena quantidade pode ter sido construída, mas podemos imaginar mais e mais sendo adicionado, à medida que é necessário. Estes computadores têm aspectos teóricos de especial interesse e serão chamados computadores com capacidade infinita.

A idéia de computador digital já é antiga. Charles Babbage, professor de matemática em Cambridge de 1828 a 1839, planejou uma máquina destas, chamada de "Máquina Analítica", mas nunca foi acabada. Embora Babbage tivesse as idéias essenciais, a sua máquina não teve, naquela época, uma perspectiva muito atraente. A velocidade com que seria disponibilizada, era sem dúvida mais rápida que o computador humano, mas cerca de 100 vezes mais lenta do que a máquina de Manchester, uma das mais lentas dentro das modernas máquinas. O armazenamento era meramente mecânico, usando rodas e cartões. O fato de que a Máquina Analítica de Babbage era para ser inteiramente mecânica, ajuda-nos a livrar-nos de uma superstição. Muitas vezes dá-se importância ao fato de os modernos computadores digitais serem elétricos, assim como o sistema nervoso é elétrico. Visto que a máquina de Babbage não era elétrica e visto que todos os computadores digitais são de certo modo equivalentes, notamos que este uso da eletricidade não pode ser de importância teórica. Claro que o assunto da eletricidade normalmente aparece quando o que preocupa é a velocidade no sinal. Por isso, não é de surpreender que a encontremos em ambos os contextos. No sistema nervoso o fenómeno químico é tão importante como o elétrico. Em alguns computadores, o sistema de armazenamento é maioritariamente acústico. O fato de se usar a electricidade é visto como sendo apenas uma semelhança muito superficial.

A Universalidade dos computadores digitais

Os computadores digitais que consideramos na última secção podem ser classificados entre as "máquinas de estados distintos". Estas são as máquinas que mudam a posição de um estado definido para outro através de saltos súbitos ou cliques. Estes estados são suficientemente diferentes, para que a possibilidade de confusão entre eles seja ignorada. Rigorosamente falando, não há máquinas assim. Na realidade, tudo se move de forma contínua. Mas há vários tipos de máquinas em que pode ser vantajoso pensá-las como sendo máquinas de estados distintos. Por exemplo, considerando os interruptores de um sistema de iluminação, é conveniente pensar que cada interruptor só pode estar na posição "ligado" ou "desligado". Poderão existir posições intermédias, contudo, para a maioria dos propósitos nós podemos esquecê-las. Como exemplo de uma máquina de estados distintos, podemos considerar uma roda que gira 120^o por segundo, podendo ser parada por uma alavanca que pode ser manipulada do exterior. Além disso, uma luz acende numa das posições da roda. Abstratamente, esta máquina pode ser descrita da seguinte maneira: o estado interno da máquina (o qual é descrito pela posição da roda) pode ser q1, q2, ou q3. Existe um sinal de entrada i0 ou i1 (posição da alavanca). A qualquer momento o estado interno é determinado pelo último estado e pelo sinal de entrada, de acordo com a seguinte tabela:

```
Último estado
      q1  q2  q3
--------------------------------
Entrada
      i0  q2  q3  q1
      i1  q1  q2   q3
```

O sinal de saída, a única indicação externamente visível do estado interno (a luz), é descrita pela tabela:

```
Estado  q1  q2  q3
Saída  o0  o0  o1
```

Este é um exemplo típico das máquinas de estados distintos. Elas podem ser descritas por estas tabelas, desde que tenham apenas um número finito de estados.

Poderá parecer que dando um estado inicial da máquina e os sinais de input, é sempre possível predizer todos os estados futuros. Isto faz lembrar a visão de Laplace que a partir do estado completo do universo num determinado momento,descrito pelas posições e velocidades de todas as partículas, deverá ser possível predizer todos os estados futuros. A predição que estamos considerando é contudo muito mais viável do que aquela que Laplace considerou. O sistema do "universo como um todo" é aquele em que pequenos erros nas condições iniciais podem ter mais tarde um efeito esmagador. O deslocamento de um único elétron por um bilionésimo de centímetro num determinado momento, pode fazer a diferença entre um homem ser morto por uma avalanche um ano mais tarde, ou escapar. Esta é uma propriedade essencial dos sistemas mecânicos, os quais chamámos "máquinas de estados distintos", em que este fenômeno não ocorre. Mesmo quando consideramos as máquinas físicas atuais em vez das máquinas idealizadas, um conhecimento razoavelmente preciso do estado num determinado momento fornece um conhecimento razoavelmente preciso nos passos posteriores.

Como mencionamos anteriormente, os computadores digitais enquadram-se na classe das máquinas de estados distintos. Contudo, o número de estado dos quais uma máquina tem capacidade, é normalmente muito extenso. Por exemplo, o número para a máquina que agora trabalha em Manchester é de cerca de $2^{165,000}$, isto é, cerca de 10^{50000}. Compare isto com o nosso exemplo da roda acima descrito, o qual tinha três estados. Não é

 ©88-09, Pedro Kantek

difícil perceber porque o número de estados deve ser tão imenso. O computador inclui um armazenamento correspondente ao papel utilizado pelo computador humano. Deve ser possível escrever no armazenamento qualquer uma das combinações de símbolos podem ter sido escritas no papel. Para simplificar, suponhamos que apenas dígitos de 0 a 9 são usados como símbolos. As variações na caligrafia são ignoradas. Suponhamos que é fornecido ao computador 100 folhas de papel, cada uma contendo 50 linhas, onde cada linha tem espaço para 30 dígitos. Então, o número de estados é $10^{100\times50\times30}$, ou seja, $10^{150,000}$. Isto é mais ou menos o número de estados de três máquinas de Manchester juntas. O logaritmo de base dois do número de estados, é habitualmente chamado de "capacidade de armazenamento" da máquina. Assim, a máquina de Manchester tem uma capacidade de armazenamento de cerca de 165,000 e a máquina da roda do nosso exemplo anterior tem 1,6. Se duas máquinas são colocadas juntas, as suas capacidades devem ser adicionadas para obter a capacidade da máquina resultante. Isto conduz-nos à possibilidade de afirmações como: "A máquina de Manchester contem 64 pistas magnéticas, cada uma com a capacidade de 2560, 8 tubos electrônicos com a capacidade de 1280. Os diversos armazenamentos atingem cerca de 300, fazendo um total de 174,380."

Dada a tabela correspondente a uma máquina de estados distintos, é possível predizer o que ela fará. Não há razão para que esta estimativa não seja levada a cabo por meio de um computador digital. Contando que pode ser levado a cabo suficientemente rápido, o computador digital pode imitar o comportamento de qualquer máquina de estados distintos. O jogo da imitação pode então ser jogado com a máquina em questão (como B) e o computador digital imitador (como A) e o interrogador seria incapaz de distingui-los. Claro que o computador digital deve ter uma capacidade de armazenamento adequada, assim como trabalhar suficientemente rápido. Além disso, deve ser programado de novo para cada nova máquina que deseje imitar.

Esta propriedade especial dos computadores digitais, que é o fato de eles imitarem qualquer máquina de estados distintos, é descrita pela afirmação de que são máquinas universais. Sem considerarmos o fator velocidade, podemos dizer que a existência de máquinas com esta propriedade tem uma importante conseqüência, que é o fato de ser desnecessário desenhar varias novas máquinas para fazer vários processos de computação. Pode ser tudo feito com apenas um computador digital, programado apropriadamente para cada caso. Veremos que uma conseqüência disto é que todos os computadores digitais são de certa maneira equivalentes. Poderemos considerar novamente o ponto em relevo no fim do capítulo 3. Foi sugerido, a título de experiência, que a questão "Podem as máquinas pensar?" deveria ser substituída por "Existirá algum computador digital imaginário, o qual terá um bom desempenho no jogo da imitação"?. Se desejarmos, poderemos generalizar e perguntar: "Existirá alguma máquina de estados distintos a qual terá um bom desempenho?". Mas, na perspectiva da propriedade da universalidade, nós vemos que cada uma destas questões é equivalente à seguinte: "Vamos fixar a nossa atenção num computador digital específico C. É verdade que alterando este computador para ter um armazenamento adequado, aumentando convenientemente a sua velocidade de ação e fornecendo-lhe o programa apropriado, C pode ser criado para jogar satisfatoriamente o papel de A no jogo da imitação e o papel de B desempenhado por um humano?"

Visões Contrárias à questão fundamental

A objeção teológica

"Pensar é uma função da alma imortal do Homem. Deus deu uma alma imortal para cada homem e mulher, mas não a nenhum outro animal ou máquina. Portanto, nenhum animal ou máquina pode pensar."

Turing não aceita nenhum ponto desta argumentação, contudo, tenta dar uma resposta a esta visão.

Ele acredita que este argumento seria mais convincente se os animais fossem classificados junto com os homens, pois a seu ver, existe uma diferença muito maior entre o inanimado e o animado do que entre o homem e os outros animais.

Turing considera que por essa dificuldade de as pessoas aceitarem que Deus pode conferir uma alma aos animais, ainda lhes é mais difícil "engolir" que Deus pode conferir uma alma às máquinas

Para Turing que os argumentos teológicos não lhe dizem muito, visto que no passado Galileu e Copérnico foram acusados pela igreja, ao serem utilizados os textos bíblicos de Josué 10:13 e Salmos 5. que dizem respectivamente: "E o sol deteve-se e a lua parou..." e "Ele lançou os fundamentos da terra para que não se abale em tempo algum"

Objeção das "cabeças na areia"

Segundo esta objeção, as conseqüências de uma máquina pensar, seriam pavorosas. Por isso, os defensores desta objeção dizem que esperam e acreditam que tal nunca venha a acontecer.

Turing argumenta e de alguma forma ironiza, dizendo que desejamos acreditar que o Homem é de alguma forma especial, mesmo que não seja superior ao resto da criação. Turing considera que este argumento está de alguma forma ligado ao argumento anterior e não desenvolve muito a sua argumentação.

Objeção matemática

Segundo a objeção matemática, existem vários resultados da lógica matemática que podem ser usados para demonstrar que há limitações no potencial das máquinas de estados distintos.

O mais conhecido é o teorema de Gödel, que demonstra que em qualquer sistema lógico suficientemente poderoso podem ser formuladas proposições que não podem ser demonstradas nem refutadas dentro do sistema, a menos que o próprio sistema seja contraditório. Contudo, existem outros, como por exemplo de Church, Kleene, Rosser e do próprio Turing, sendo este último mais apropriado para considerar, visto que se refere diretamente às máquinas. Ou seja, o próprio Turing demonstrou que há limitações no potencial das máquinas de estados discretos, através por exemplo do problema da paragem (parada).

Se uma máquina for investida para dar respostas às perguntas do jogo da imitação, haverá perguntas à qual dará uma resposta errada, ou simplesmente não responde, apesar de ter tempo suficiente.

Turing argumenta que não há provas que o intelecto humano não sofra das mesmas limitações de uma máquina. Se uma máquina não responder ou der uma resposta errada, pode um ser humano sentir-se superior?

No fundo, nós também somos falíveis, por isso não nos devíamos sentir assim tão satisfeitos com as evidências de falha por parte das máquinas. Poderá haver homens mais "espertos" que algumas máquinas, assim como haverá máquinas mais "espertas" que alguns homens. Assim, a nossa superioridade em relação às máquinas só devia ser sentida ocasionalmente, quando conseguimos um pequeno triunfo em relação a alguma.

O argumento da consciência

Turing cita o professor Jefferson que argumenta que uma máquina não é equivalente a um cérebro, a menos que seja capaz de escrever um soneto por causa das emoções

 ©88-09, Pedro Kantek

e sentimentos sentidos e ter consciência que o escreveu. Por outras palavras, se uma máquina não é consciente, então não pode pensar.

Turing declara que esta visão provavelmente nega a validade do jogo da imitação. A única forma que então teríamos para descobrir se uma máquina pensa é ser a própria máquina e sentirmo-nos pensando, ou seja, de acordo com esta visão, a única maneira de sabermos se um determinado homem pensa, é ser esse homem, o que Turing considera um ponto de vista SOLIPSISTA. (filosofia segundo a qual a única realidade do universo é o *eu*).

De forma a persuadir que o jogo da imitação é um bom teste, Turing dá o exemplo de um jogo, conhecido como "viva voz", que é usado para descobrir se alguém realmente compreendeu algo, ou se aprendeu de uma forma estilo "papagaio". Neste exemplo, a máquina responde a perguntas que envolvem as escolha de uma metáfora num soneto, de maneira digna de um crítico de poesia.

Turing conclui também dizendo que é sensível ao problema da consciência, não querendo dar a impressão que pensa que não há mistérios em relação a este assunto. Mas acredita que há um certo paradoxo na tentativa de localizá-la e que isso não é relevante para a nossa questão.

Argumentos a partir de várias incapacidades

Este argumento defende que uma máquina pode fazer tudo, menos X.

X poderá ser: Ser amável, bonito, amigável, ter iniciativa, ter senso de humor, fazer erros, apaixonar-se, apreciar morangos com creme, ser o assunto do seu próprio pensamento, etc.

O primeiro argumento de Turing relativo a esta visão é que esta argumentação não tem fundamento, pois a idéia das pessoas relativamente ao que uma máquina pode fazer, é generalizada através de uma indução científica daquilo que elas já viram.

Contudo, Turing pega nalgumas das incapacidades atribuídas às máquinas e refuta.

Relativamente à incapacidade de cometer erros, Turing utiliza como exemplo jogo da imitação. Poder-se-á pensar que o interrogador fará um série de problemas aritméticos e a máquina será desmascarada por causa da sua precisão. Contudo, uma máquina (programada para jogar) não tentará dar sempre as respostas corretas às perguntas aritméticas. Ela deliberadamente introduzirá erros para confundir o interrogador. Turing argumenta que esta crítica está fundamentada numa confusão entre dois tipos de erros: "Erros de funcionamento" e "Erros de conclusão". É verdade que as máquinas (por definição as máquinas abstratas que estamos aqui discutindo) não podem cometer estes "Erros de Funcionamento".

Contudo, as máquinas podem ser programadas para fazerem erros ocasionais para imitar o comportamento humano, como por exemplo para imitar uma pessoa escrevendo no computador. Quando estes erros ocorrem podemos dizer que foi cometido um "Erro de Conclusão". Por outro lado, se as máquinas chegarem a conclusões por indução científica, isso poderá levá-las a "Erros de Conclusão", visto que a indução não é infalível.

Relativamente à incapacidade de uma máquina ser o assunto do seu próprio pensamento, Turing argumenta que num certo sentido, as máquinas podem ser assunto do seu próprio interesse. Pode ser usado para ajudar a completar os seus próprios programas ou a predizer os efeitos das alterações na sua própria estrutura. Pela observação dos resultados do seu próprio comportamento ela pode modificar os seus programas para conseguir algum propósito mais efetivamente.

Turing conclui dizendo que a crítica analisada neste ponto é muitas vezes disfarçadas dos argumentos do ponto de vista anterior sobre as consciência.

Objeção de Lady Lovelace

Lady Lovelace, num discurso sobre a máquina Analítica de Charles Babbage, cujo desenho incorpora todas as características que a fazem equivalente a um computador digital universal, faz a seguinte afirmação. "O engenho analítico não tem pretensões de originar nada. Ele pode fazer tudo o que nós sabemos mandá-lo fazer" ou seja, aquilo que nós saibamos programar.

Turing argumenta uma variante da objeção de Lovelace que diz que uma máquina "nunca pode fazer nada realmente novo".

Turing questiona sobre quem pode ter a certeza que o trabalho que faz é "original" e não foi criado a partir do crescimento de uma semente plantada nele pelos ensinos que obteve, ou sendo o efeito de princípios já bem conhecidos?

Uma melhor variante desta objeção é que uma máquina "nunca nos pode surpreender". Turing argumenta que as máquinas o surpreendem com muita freqüência, porque produzem resultados corretos que estão longe do que ele esperava através das suas estimativas. Ele refere que a surpresa é algo que tem mais a ver com um ato criativo da parte da mente que a detecta, do que algo que é originado na máquina (no homem, etc).

Argumento sobre a continuidade no sistema nervoso

Este argumento refere-se ao fato de que o sistema nervoso não é uma máquina de estados distinto e visto que um pequeno erro ao medir o impulso de entrada de um neurónio pode fazer uma grande diferença para o tamanho do impulso de saída de um neurónio, não poderemos imitar o sistema nervoso como um sistema de estados distintos.

Turing concorda com o facto de haver uma diferença entre uma máquina de estados distintos e uma máquina contínua. Mas ele argumenta que o interrogador no jogo da imitação não pode explorar esta diferença para tirar vantagem.

Turing dá o exemplo de um analisador diferencial (que é uma máquina usada para alguns tipos de cálculos, a qual não é do tipo da máquina de estados distintos.) e diz que embora seja impossível predizer exatamente que resposta o analisador diferencial dará a um problema, como por exemplo estimar o valor de x, o computador digital pode dar uma resposta probabilística que será muito difícil para o interrogador distinguir da resposta do analisador diferencial.

O argumento da informalidade do Comportamento

O argumento é que o comportamento humano não pode ser representado por um conjunto de regras de conduta que determina cada ação humana. Por isso, os seres humanos não podem ser máquinas.

O argumento é que dado um conjunto fixo de regras, podemos sempre imaginar uma situação para a qual nenhuma regra é aplicável. Ele dá o seguinte exemplo: Podemos ter um regra que diga que devemos parar quando virmos um sinal vermelho e para avançar se virmos um sinal verde. Mas se ambos aparecerem ao mesmo tempo ? Como decidir? Talvez decidissemos que o mais seguro é parar. Mas esta decisão poderia trazer problemas. Não é por isso possível criar regras de conduta para abranger todas as eventualidades.

Através desta impossibilidade de descrever o comportamento humano num conjunto de regras de conduta, os proponentes desta argumentação concluem que os homens não podem ser máquinas.

Turing concorda que é impossível inventar um conjunto de regras que irão governar o comportamento de uma pessoa em todas as situações concebíveis. Contudo considera que existe uma confusão entre regras de conduta e leis de comportamento. Por "regras de conduta" ele quer dizer normas do tipo: "Parar quando vir uma luz vermelha", sobre

 ©88-09, Pedro Kantek

as quais podemos agir e das quais temos consciência. Por "leis de comportamento" Turing refere-se às leis da natureza aplicadas ao corpo humano, como por exemplo: "Se beliscares alguém, ele vai gritar".

Turing considera que é mais difícil nos convencermos de que não somos governados por leis de comportamento, do que por regras de conduta.

Argumento da Percepção Extra-Sensorial

Turing considera que há evidências estatísticas para a percepção Extra-Sensorial. Ele também considera que se há comunicação telepática entre o interrogador e o computador, o interrogador pode fazer uma identificação correta, visto que a máquina não tem poderes telepáticos.

A resposta de Turing é para colocar os participantes numa sala "à prova de telepatia", de forma que isso não interfira no jogo.

Continuação - Reações ao teste de Turing

Gunderson (1964), num dos primeiros comentários ao artigo de Turing, questiona a própria existência de imitação (por parte da máquina) no "jogo da imitação". O seu ponto é que, lá por um artefato substituir um humano no seu papel no jogo, isso não significa que esse artefato esteja a imitar o que quer que seja.

Para Purtill (1971), o jogo da imitação não é mais do que uma batalha entre o interrogador humano e o humano que programou a máquina.

Miller (1973) terá sido um dos primeiros a criticar o fato de o jogo da imitação assentar numa concepção antropomórfica de inteligência: a inteligência que ele pode detectar é a inteligência adaptada aos objetivos e à cultura dos humanos, sendo desejável que se permita a um marciano ou a uma máquina exibir inteligência através de comportamentos adaptados à prossecução dos seus objetivos próprios.

Moor (1976) dá uma interpretação indutiva do jogo da imitação: o jogo não nos dá uma definição operacional de inteligência e não constitui uma condição necessária nem suficiente para o reconhecimento de inteligência da máquina. A sua utilidade consiste em fornecer elementos para uma inferência indutiva com uma conclusão favorável à hipótese da inteligência das máquinas, numa apreciação reversível: podemos ter aceite que uma máquina é inteligente com base nos resultados do jogo da imitação, mas revermos posteriormente essa apreciação com base noutros elementos. Moor concorda, no entanto, que o jogo proposto por Turing permite testar muitos aspectos da inteligência.

Para Stalker (1978), a atribuição de pensamento a uma máquina em certas condições é uma tentativa de explicar o comportamento da máquina nessa situação: existindo explicações puramente mecânicas - que são mais "econômicas"do que as explicações mentalistas - elas são preferíveis.

A inteligência de uma torradeira

Ned Block (1981) ataca o teste de Turing por ele constituir uma abordagem behaviorista à inteligência. O seu objectivo é opôr o psicologismo ao behaviorismo. A tese psicologista que Block pretende defender é a seguinte: que um comportamento seja ou não seja inteligente depende da natureza do processamento interno de informação que o produz; não bastará nunca o comportamento para indicar se um sistema é ou não inteligente. O erro do teste de Turing é querer decidir a questão da inteligência apenas pelo aspecto exterior do comportamento.

Tratemos de dar um esquema do argumento de Block. Block fornece um neo-teste de Turing que mantém as características essenciais do original e, como ele, apenas atende ao comportamento exterior; vai mostrar que uma máquina que reconhecemos (por análise

dos seus mecanismos internos) como não sendo de todo inteligente, pode passar esse neo-teste de Turing; vai daí concluir que o neo-teste de Turing (bem como o original) deve ser declarado imprestável como critério de atribuição de inteligência.

Mantendo-se, como o jogo da imitação da versão original, no domínio do comportamento "uso de uma linguagem natural por escrito", esse neo-teste de Turing para a inteligência assenta nesta noção: a inteligência é a capacidade para dar sequência a um diálogo sensato. Qualquer diálogo mantido durante um certo intervalo de tempo finito é uma seqüência finita de frases, formadas por um número finito de palavras. Chamemos "diálogo sensato"a qualquer dessas seqüências em que cada fala de um dos interlocutores seria reconhecida por um humano como uma continuação razoável das falas anteriores. O número de "diálogos sensatos"com uma duração determinada (por exemplo, os cinco minutos que Turing propõe) que é possível formar combinatoriamente, é um número finito - embora muito grande. Uma máquina podia ser fornecida com todos esses "diálogos sensatos"e ser programada para responder a um ser humano apenas por consulta da "base de diálogos sensatos". Assim, qualquer frase do humano será a frase A de um dos diálogos armazenados e a máquina "responderá"com uma das possíveis frases B (a frase B de qualquer dos diálogos que começam com esse A). O humano seguirá necessariamente com uma frase C de um dos diálogos que começam por aquela sequência AB e a máquina responderá com a frase D de qualquer dos diálogos que começam por essa sequência ABC. E assim sucessivamente, até se esgotar o tempo.

Essa máquina, na expressão de Block, "terá a inteligência de uma torradeira"e, no entanto, passará o neo-teste de Turing (dará seqüência a um diálogo sensato). Logo, o neo-teste de Turing (tal como o original) não serve como critério para determinar a presença de inteligência. Porquê? Porque só atende ao comportamento exterior.

Testes alternativos

Torna-se interessante acompanhar a forma que tomaram a partir de certa altura as críticas ao jogo da imitação: testes alternativos ao teste de Turing clássico. Vários autores seguiram esse método e essa estratégia contribuiu para a idéia de que o teste de Turing pretendia simplificar "à força"um problema complexo. Vejamos exemplos desses testes alternativos.

Harnad (1989) propõe o Teste de Turing Total, em que já não se aceita que a máquina "esconda"a sua implementação física atrás da comunicação por teletipo, antes se exige que a revele: a máquina submetida ao Teste de Turing Total será um robot envolvendo todos os aspectos que podem contribuir para o desempenho complexo de um sistema com mente, incluindo competências sensoriais e motoras e não fugindo à expressividade da aparência física.

Watt (1996) propõe o Teste de Turing Invertido. Watt parte da ideia de que o aspecto central do jogo da imitação proposto por Turing é a tendência natural dos humanos para atribuir estados mentais aos outros humanos e a si próprios - e também às máquinas. (É por isso que, p.ex., falamos de duas imagens na tela do computador dizendo "ele vai comer o outro".) Essa "psicologia ingênua", que nos é necessária para prever e compreender os comportamentos em sociedades complexas, é responsável pelo enviesamento da tendência do interrogador no teste de Turing para atribuir mentalidade à máquina. A psicologia do observador/interrogador humano seria mais importante do que o próprio desempenho da máquina candidata. Nessa base, Watt propõe o Teste de Turing Invertido, em que a máquina terá de desempenhar o papel do observador. A máquina terá de provar ser dotada de "psicologia ingênua"sendo incapaz de distinguir entre dois humanos, sendo também incapaz de distinguir entre um humano e uma máquina que tenha passado o teste de Turing clássico - mas sendo capaz de distinguir entre um humano e uma máquina que não tenha passado o teste clássico.

 ©88-09, Pedro Kantek

O teste da gaivota

Robert French (1990) considera que o teste de Turing não é um teste de inteligência em geral, mas apenas um teste de inteligência humana. Para ilustrar o seu ponto - a dependência do teste de Turing face ao contexto cultural dos seus intervenientes humanos - exemplifica com o "teste da gaivota".

Seja uma pequena ilha nórdica onde o único objeto voador conhecido é a gaivota. Os habitantes da ilha, convencidos de que pode haver outros objetos voadores, estabelecem o seguinte teste para determinar o que é ser voador: o registo do comportamento do candidato a "voador"num monitor de vídeo tem de ser indistinguível, aos olhos dos ilhéus, do registo do comportamento de uma gaivota no mesmo monitor. Torna-se óbvio que há inúmeros objetos que nós consideramos voadores (desde outras aves até morcegos, abelhas ou helicópteros e aviões) mas que não passarão o "teste da gaivota- porque o teste é (para usar a terminologia de Block) chauvinista em relação ao voar focado nas gaivotas, do mesmo modo que o teste de Turing é chauvinista em relação à inteligência focada nos humanos.

O teste de redação

Collins (1997) propõe uma variante do teste de Turing cuja originalidade consiste em, em lugar de procurar ser mais exigente do que a versão clássica, constituir um sub-teste de Turing - no sentido em que considera que uma exigência mais restrita é representativa da exigência original e, por isso, é suficiente para obter os mesmos resultados.

Na análise de Collins, o que está em causa são as poderosas modalidades de aprendizagem que os humanos adotam para se adaptarem ao seu meio, incluindo ao seu meio social - residindo aí o cerne da desvantagem comparativa das máquinas. Enquanto os humanos, ao interagirem, interpretam o que acontece na interação e usam essa interpretação para corrigir deficiências na leitura dos dados que lhes chegam (posso compreender frases mal formuladas, seja de forma flagrante seja de forma sutil, recorrendo a elementos de contexto) - os computadores não têm essa capacidade (só funcionam bem se lhes fornecerem dados ou comandos de forma estritamente identificável: o uso de ícones na tela do meu computador serve para limitar drasticamente o que eu posso "dizer-lhe"). Essa diferença, que Collins designa por "assimetria interpretativa", parece-lhe tão fundamental que pode ser captada por uma variante limitada do teste de Turing sem perder o poder discriminatório que se lhe exige. Esse sub-teste de Turing seria um Teste de Redação.

O Teste de Redação se assenta na idéia de que todo o poder das nossas capacidades interpretativas pode ser adequadamente representado pelo seu uso num contexto lingüístico e, aí, distinguir-nos da máquina. O teste consistiria apenas em exigir que certas frases em linguagem natural, apresentadas com certas incorreções, sejam apropriadamente corrigidas pelo sistema candidato à qualificação de inteligente.

Testar a "mãe natureza"

Schweiser (1998) considera que as formas conhecidas de teste de Turing, incluindo a versão robótica de Harnad, são insuficientes. Os humanos atribuem inteligência a outros humanos com base na observação do comportamento (mesmo na falta de uma boa teoria da inteligência humana) porque têm um registo histórico muito rico acerca da espécie. O que se passa face a um humano individual é que a nossa atribuição de inteligência é basicamente uma identificação de um espécime (aquele indivíduo) como pertencente a um tipo (a espécie humana). Assim, a atribuição de inteligência a um tipo cognitivo diferente (robots) requer a produção de um registo histórico das realizações dessa "espécie"que lhe garanta a mesmo tratamento: não basta que os robots falem uma linguagem

ou joguem xadrez, eles terão que desenvolver uma linguagem e inventar um jogo como o xadrez. Assim, o teste de inteligência tem primeiro de ser passado ao nível do tipo e só depois (e nessa base) aplicado ao nível dos espécimes.

Alguns autores têm analisado a questão do ponto de vista dos aspectos sociais da inteligência, muitas vezes indicando que a adaptação, a aprendizagem, a comunicação, a cooperação - são aspectos importantes da inteligência que desaparecem na versão original do teste de Turing. Deste ponto de vista, uma das propostas mais radicais será certamente a de Barresi (1987), com o seu "Cyberiad Test". John Barresi considera que o comportamento inteligente é aquele que permite à sociedade sobreviver e que o árbitro desse "jogo"é a "mãe natureza". Por tanto, o seu teste, aplicado a uma sociedade de humanos artificiais, seria passado com sucesso se essa sociedade continuasse a sua evolução sócio-cultural sem se desintegrar durante uma história suficientemente longa - digamos, alguns milhões de anos.

Lagoa dos patos

 ©88-09, Pedro Kantek

Capítulo 3

Uma visão moderna da IA

3.1 Introdução

O homem deu a si o nome de *homo sapiens*. Só este fato já mostra a valoração que damos a nossas habilidades mentais. A Inteligência artificial (IA) busca compreender entidades inteligentes. Em conseqüência ao estudar IA queremos saber mais sobre nós mesmos. Até aqui, a IA compartilha o objeto de interesse com a filosofia e a psicologia. Mais adiante, a IA quer compreender e também **construir** entidades inteligentes. Uma dificuldade/facilidade (depende do ponto de vista), é que temos um bom modelo para isso: basta se olhar no espelho.

Tais entidades têm interesse para nós, pois tem valor próprio. O tamanho do mercado da informática fala sobre isso. Não obstante sua infância, a IA já criou produtos de transcendência e surpreendentes.

Junto com a genética humana, a IA é o segundo campo em que todo cientista, de qualquer área, gostaria de atuar. Por exemplo, na física as questões fundamentais foram propostas por Galileu, Newton e Einstein. Qualquer novato, para contribuir terá que estudar anos.

Na matemática ocorre algo parecido. O recente episódio da prova do *Último Teorema de Fermat* levou quase 40 anos da vida de Andrew Willes. Os 7 problemas do milênio, propostos agora no século XXI, com 1 milhão de dólares de prêmio para quem conseguir resolver qualquer um deles, falam disso também: só para entender a formulação de cada problema, o sujeito (poucos no mundo) gastarão anos, veja em www.claymath.org.

Ao contrário, na IA, há muitas (muitas!) áreas onde é possível sentir-se como Einstein ou como Fermat.

Há 2000 anos pelo menos, a filosofia especula e quer saber como se aprende, recorda e raciocina. A chegada do computador nos anos 50, permite passar da especulação e da instrospecção para a teorização, seguida da experimentação.

Uma lição que ficou é que a IA é muito mais complicada do que se achou a princípio. Este erro de avaliação pode ser exemplificado pela dificuldade de dimensionar uma nuvem estando dentro dela.

A IA é um convite a profissionais de todas as outras áreas para que, usando seu ferramental possam sistematizar e automatizar seu trabalho intelectual. Com isso, aos técnicos da IA, abriu-se um campo sem limites, genuinamente universal.

3.1.1 Inteligência Artificial

Existe dificuldade básica que é a de definir o objeto maior deste estudo. Embora o termo "inteligência artificial" e a sigla "IA" tenham se firmado como uma importante área de

pesquisa nas ciências cognitivas e da informação, não há consenso entre os especialistas a respeito do que seja IA. As dificuldades estão nos dois pontos: as palavras artificial e, principalmente, a palavra inteligência.

Minsky diz que

> a máquina está a ser inteligente quando a tarefa que ela executa necessita de inteligência quando feita por seres humanos

Já Kvitca afirma que

> la IA es la ciencia que estudia la forma de diseñar programas de computación que exhiban características que comunmente asociamos con el comportamento humano inteligente

Note-se que em ambas as definições, e em geral em todas elas, temos uma associação ao comportamento humano. Isso nos remete à questão do que é a inteligência humana, que também não tem resposta satisfatória. Outros autores preferem uma definição recursiva como Winston que diz "artificial intelligence is the study of ideas that enable computers to be intelligent". [Win84].

Uma boa definição é dada por Elaine Rich, que diz IA é o estudo de como fazer os computadores realizarem coisas que, no momento, as pessoas fazem melhor. Esta definição, além de fugir da questão espinhosa da inteligência, ainda dá idéia de movimento e de imprecisão, algo que é importante em IA. De fato, as fronteiras deste ramo do conhecimento não são fixas, pois à medida em que os mecanismos que governam as ações ditas "inteligentes" passam a ser compreendidos, parece que aquelas ações deixam de ser inteligentes. Isto conduz-nos à idéia apresentada por Larry Tesler segundo a qual a inteligência artificial é tudo aquilo que ainda não foi feito, atribuindo aos técnicos da IA uma situação de eternos vencidos [Mic85, pág. 15]. Outro ponto que sugere movimento, é que ramos da IA quando tomam impulso e desenvolvimento passam a se constituir em ciências autônomas, como por exemplo ocorreu com a robótica ou com a visão computacional. Alguém mais inspirado, chamou a IA de "saturniana", já que este personagem da mitologia grega, mata os seus filhos.

3.1.2 Do que se ocupa a IA

Infelizmente ainda nos falta uma lei geral ou unificadora dentro da IA (e até da ciência da computação) embora este seja um dos ramos onde há maior pesquisa nos tempos atuais. Enquanto tal idéia unificadora não vem (se é que vem), há necessidade de se contentar com uma lista exaustiva dos assuntos de que se ocupa a IA. Praticamente todos os estudiosos do tema incluem suas relações, como por exemplo, Barr que diz "understanding language, learning, reasoning, solving problems and so on" [Bar81a]. Uma lista um pouco mais extensa está em [Win84] que sugere "description matching and goal reduction; exploiting natural constraints; exploring alternatives; control metaphors; problem solving paradigms; logic and theorem proving; representing commonsense knowledge; language understanding; image understanding; learning class descriptions from samples; learning rules from experience".

3.1.3 O que é inteligente ?

Parte da dificuldade de caracterizar a IA é a dificuldade de definir inteligência. Alguns dirão impossibilidade, é uma questão aberta. Alan Turing (1912-1954), possivelmente o maior cientista da computação, saiu-se pela tangente: Em seu trabalho de 1950, Computing Machinery and Intelligence, ele propôs um teste para detectar um comportamento inteligente. Ainda não tinha o seu nome, foi-lhe dado depois.

As habilidades necessárias para um programa passar no TT são

- processamento em linguagem natural
- representação do conhecimento
- raciocínio automático
- aprendizagem

Posteriormente, propôs-se o chamado Teste Total de Turing (TTT) no qual o dispositivo em teste não é apenas software, mas sim hardware+software carregado em um robot humanoide. Neste caso, a iteração se dá através de duas formas: Um sinal de vídeo, pelo qual se pode ver e ouvir os dois competidores e um guichet pelo qual se pode ceder e receber objetos para os ou dos competidores

Para o TTT, além daquelas habilidades acima citadas, são também necessárias a

- habilidade da visão
- capacidade robótica

Se há dúvidas quanto a possibilidade de algum programa passar no TT, com maior razão há muitas mais dúvidas quanto à aprovação num TTT.

Uma observação que se deve fazer aqui é quanto ao programa ELIZA de Joseph Weizenbaum (1965), o qual simula uma sessão de psicanalise. O programa usa regras regras triviais para iteragir e, a despeito disso, causou algum impacto quando veio à luz. Em 1994, Mauldin construiu uma versão bem mais sofisticada do ELIZA, agora chamado JULIA.

3.1.4 Captchas

> Às vezes você vai a um site abrir uma nova conta de email grátis num provedor e, depois de digitar suas informações obrigatórias, aparece uma quadrinho contendo letras e números embaralhados, distorcidos, com umas linhas no meio e com cores de fundo que confundem tanto a nossa vista que quase não conseguimos entender o que está escrito. E ainda pedem para digitarmos os caracteres num campo logo abaixo. Pois bem, aí está. Estas figurinhas confusas são chamadas captchas. E para quê servem?
>
> Acontece que existem programinhas maliciosos, os "bots"(vem de robots, robôs em inglês), que automatizam operações repetitivas. Sem os captchas, esses bots seriam capazes de simular um humano digitando informações aleatórias de modo a criar centenas ou até milhares de contas novas de email em poucas horas. Um captcha apresenta caracteres tão distorcidos que quase nenhum programa seria capaz de interpretá-los e decifrá-los de maneira automática. Teoricamente, só mesmo um humano conseguiria entendê-lo e digitar a seqüência correta, comprovando que é um humano e não um bot.
>
> Captchas também são usados para evitar a ação de bots em votações online, assinaturas em listas e em respostas a pedidos de confirmação de identidade, no caso de ferramentas ativas anti-spam. Até em consultas a CPF no site da Receita já se usam captchas. A rigor, um captcha na verdade é um teste de Turing reverso, pois é um computador que verifica se quem está do outro lado é um humano ou um "bot".
>
> O termo "captcha"foi cunhado no ano 2000 por Luis von Ahn, Manuel Blum e Nicholas J. Hopper da Universidade Carnegie Mellon, e por John Langford da IBM. Mas a idéia é anterior, de 1997, quando Andrei Broder e seus colegas de trabalho no AltaVista usaram o primeiro captcha para evitar que

 ©88-09, Pedro Kantek

bots inserissem URL's em sua máquina de busca. Tentando evitar ataques de bots que utilizassem técnicas de OCR (optical character recognition = reconhecimento óptico de caracteres), eles caíram de boca no manual de um scanner Brother e desenharam figuras com todos os defeitos que a publicação ensinava a evitar, caso se desejasse um bom reconhecimento de escrita.

Hoje em dia, encontrar um captcha não é nada difícil. Só que já tem uma rapaziada derrotando os captchas. Acredite, tem sim. São brilhantes esses caras. Eles estão construindo bots que capturam um captcha original em um site X, apresentam-no a um humano num site diferente Y, em outro contexto, registram a resposta da criatura e submetem-na de volta ao campo digitado no site X de onde capturaram a figura, transpondo a barreira de segurança e ganhando acesso a seja lá o que for que estão querendo acessar. Em geral, esses gênios usam como site Y páginas de pornografia, que os tarados virtuais acessam em busca de material para seus devaneios. Assim, fingindo que é uma barreira real, o bot apresenta ao ávido internauta o captcha capturado no site X. O camarada vai lá e digita sua interpretação da figura. O bot pega os caracteres digitados no site pornô Y, que está sob seu controle, e "digita"os mesmos caracteres no site X. E pronto, mais uma vitória da inteligência a serviço do mal.

Existem outros métodos engenhosos para burlar captchas, como o uso de "sweatshops"de operadores humanos, a utilização de algoritmos cabeludos e o reaproveitamento da "session-ID"de um captcha conhecido. Mas o espaço aqui é pouco e não posso me alongar tanto. Estou certo, porém, que a leitora é boa de fuxicação e vai dar uma olhadela lá no verbete apropriado da Wikipedia. Pena que, por enquanto, só tem em inglês.

o texto sobre catpchas foi extraído de http://www.mail-archive.com/goldenlist@yahoogroups.com/msg00908.html, escrito por Carlos Alberto Teixeira, em 15/12/06

Uma sessão de Eliza

Esta sessão foi tirada de *ELIZA - A Computer Program for the Study of Natural Language Communication between Man and Machine, da Communications of ACM, Jan 1965, pags 36 a 45* devidamente traduzida. As frases ditas pelo humano estão em letra minúscula e as respostas do ELIZA estão em maiúsculo.

```
Os homens são todos semelhantes.
DE QUE FORMA
Estão sempre a aborrecer-nos com isto ou com aquilo.
PODE DAR UM EXEMPLO ESPECÍFICO
Bem, o meu namorado obrigou-me a vir AQUI.
O SEU NAMORADO OBRIGOU-A A VIR AQUI
Diz que ando constantemente deprimida.
LAMENTO SABER QUE ANDA DEPRIMIDA
É verdade.  Sinto-me infeliz.
ACHA QUE A SUA VINDA AQUI A AJUDA A SENTIR-SE MENOS INFELIZ
Preciso de ajuda, disso não tenho dúvidas.
O QUE SIGNIFICARIA PARA VOCÊ CONSEGUIR UMA AJUDA
Talvez conseguisse aprender a dar-me melhor com a minha mãe.
FALE-ME DA SUA FAMÍLIA
A minha mãe cuida de mim.
QUEM MAIS NA SUA FAMÍLIA CUIDA DE VOCÊ
```

```
O meu pai.
O SEU PAI
De certa forma, faz-me lembrar o meu pai.
QUE SEMELHANÇA VE
Não é tão agressivo, mas acho que não me quer deixar ver isso.
O QUE A LEVA A PENSAR QUE EU NÃO SOU MUITO AGRESSIVO
Não discute comigo.
PORQUE PENSA QUE NÃO DISCUTO CONSIGO
Tem medo de mim.
AGRADA-LHE ACREDITAR QUE TENHO MEDO DE VOCÊ
O meu pai tem medo de toda a gente.
QUE MAIS LHE OCORRE QUANDO PENSA NO SEU PAI
Covardia.
ISSO ESTÁ RELACIONADO COM O FATO DE O SEU NAMORADO A TER OBRIGADO
A VIR AQUI ?
```

Em 1991, Hugh Loebner estabeleceu um prêmio de US$ 100.000 para o autor do primeiro programa que seja aprovado em um teste de Turing irrestrito. A competição tem lugar anualmente, com prêmios menores para os melhores programas em testes restritos.

3.1.5 Enfoques do pensamento

Existem pelo menos [1] duas maneiras de estudar o processo do pensamento.

modelo cognitivo Pensar é ... pensar como os humanos. Neste caso, precisamos imitar o homem. A próxima pergunta é *como pensam os homens ?* ou ainda *como pensamos ?*

modelo racional Pensar é ... usar as leis do pensamento. Este parece ser a melhor abordagem. Supostamente, temos apenas o computador para fazer experimentos.

No primeiro caso (cognitivo), as ferramentas que podem ser usadas são a introspecção (Difícil e complicada, como já nos ensinou Freud) ou então os experimentos psicológicos, que também são complicados e nem sempre conclusivos. Um exemplo desta abordagem foi o programa GSP (General Solver Problem) apresentado por Simon em 1961. O enfoque aqui não era resolver o problema, mas estudar como o programa poderia resolvê-lo vis-a-vis humanos resolvendo o mesmo problema.

Este enfoque deu origem à ciência cognitiva, que é uma ciência multidisciplinar que agrega IA e psicologia em único campo. O objetivo desta ciência é o estudo da mente humana. Algumas (poucas) universidades no mundo alocam a pesquisa de IA sob este tema.

Já o enfoque racional começa com os silogismos de Aristóteles, que por sua vez deram origem à lógica, que quando usada em IA usualmente denomina-se enfoque logicista. Duas dificuldades aparecem aqui: 1. Não é fácil representar conhecimento informal usando-se a lógica formal (onde fica a incerteza ?) e 2. Uma coisa é resolver um problema em princípio e outra é resolvê-lo de verdade. Ocorre aqui o que foi chamado de *maldição da dimensionalidade* fato este que gerou os chamados *toy domains* presentes em muitas aplicações de IA, mesmo hoje.

Apesar dessas dificuldades a IA começou aqui.

[1] O pensamento parece uma coisa à toa, mas como é que a gente voa, quando começa a pensar... Caetano Velloso ou "Reflexão sobre a reflexão": Terrível é o pensar. Eu penso tanto. E me canso tanto com meu pensamento que às vezes penso em não pensar jamais. Mas isto requer ser bem pensado Pois se penso demais Acabo despensando tudo que pensava antes E se não penso Fico pensando nisso o tempo todo. de Millor Fernandes

 ©88-09, Pedro Kantek

3.2 História da IA

Seguramente há 2500 anos, na Grécia antiga começaram a surgir as teorias do raciocínio e da aprendizagem. Mais precisamente durante algumas décadas por volta de 420 aC, um trio bacana andou produzindo as bases de toda a cultura ocidental. Fala-se de Sócrates, seu aluno Platão e o aluno deste, Aristóteles. Estes 3 abordaram a política, a matemática, a física, astronomia e muita, muita filosofia.

Posteriormente, há cerca de 400 anos, foi a vez dos matemáticos assumirem. Tudo começou com René Descartes que por volta de 1630 estabeleceu a teoria do dualismo, afirmando que o homem é constituído de corpo e mente. A diferença para os animais é de que estes têm apenas corpo. A resposta veio através de Leibniz, que em 1700 aproximadamente, afirma o contrário: Tudo é matéria, e toda a matéria está sujeita às leis do Universo, ainda que estas leis sejam desconhecidas. O nome desta doutrina é, a propósito, *materialismo*.

No fim das contas, segundo Leibniz, a mente é uma máquina que funciona como um dispositivo físico que raciocina manejando o conhecimento até ali depositado. A pergunta seguinte é *E quem botou lá esse conhecimento ?*, ou de maneira mais geral, qual a origem do conhecimento ?

Uma possível resposta vem a partir do trabalho de Bacon (1620) e Locke (1700) que dizem *"Nada existe na mente que não tenha passado antes pelos sentidos"*.

Outra, um pouco mais tortuosa, vem de Russel, já no século XX que diz *"O conhecimento advém da interrelação entre orações de observação, que entraram via sensorial"*. O nome desta doutrina: Positivismo lógico.

3.2.1 Agentes

Nos primórdios da IA, a preocupação, era a construção de programas inteligentes, capazes de perceber um estado de coisas e sugerir o melhor curso de ação. À medida em que o tempo foi passando, surgiu, já nos anos 1990, uma abordagem mais moderna de IA. Segundo esta, *IA é construir agentes inteligentes*. Assim, precisa-se definir o que seja um agente inteligente: Trata-se de algo capaz de perceber e atuar. Assim, IA é o estudo e a construção de agentes racionais. A visão antiga da IA (fazer inferências corretas) é, ou pode ser, apenas uma parte da visão mais moderna. Estudar IA pelo enfoque de agentes racionais tem vantagens:

- É mais geral do que aquele usando as leis do pensamento. A inferência é útil, mas nem sempre suficiente (e mesmo algumas vezes, não necessária).
- É mais próxima ao processo de desenvolvimento do homem. Esta característica é importante, já que em entornos complexos é impossível a racionalidade perfeita (traduzida por *sempre fazer o certo*).

Voltando ao fio histórico, surge a pergunta: qual a relação entre conhecimento e ação ?

Aristóteles já dizia "o que nos preocupa é o fim". Daí, deve-se ir aos meios que vão permitir chegar ao fim desejado. (Este curso de raciocínio recebeu o nome de *ética Nicomáquea, Aristóteles*. Este enfoque foi implementado por dois pesquisadores, Newel e Simon, num programa chamado GSP, cerca de 2300 anos depois de Aristóteles. Aqui, explora-se a diferença entre o que se tem e o que se quer.

> Quero levar meu filho no jardim de infância. Qual a diferença entre o que quero e o que tenho ? Uma distância. Que posso usar para eliminá-la ? Meu carro. Mas este não liga. Que preciso para que ele funcione ? Uma bateria nova. Onde tem baterias novas ? Em uma oficina de mecânico eletricista. Gostaria que o mecânico trocasse a minha bateria, mas ele ignora o que

> desejo. Onde está o problema ? Na comunicação. Como posso comunicar ? Usando o telefone. ...
>
> O GSP fazia este tipo de análise, classificando as coisas de acordo com sua função e em seguida passar de fins, funções requeridas e meios necessários para realizá-las. Isto constituia a heurística básica do GSP.

3.2.2 As ciências necessárias

A matemática assumiu o bastão depois da idade média. Para passar da filosofia para uma ciência formal como a matemática, necessitou-se desenvolvimentos em 3 áreas: computação, lógica e probabilidade. Alguns expoentes importantes: Al Jwarizmi, Cardano (1501-1576), Fermat, Pascal, Bernouilli, Boole, Frege, Hilbert, Gödel, Bayes, Turing, entre outros.

A psicologia, também foi importante, tendo como marco o livro *Handbook of physiological optics* publicado em 1879 por Helmholtz. Ele inaugurou a tradição existente até hoje de mesclar a IA e a ciência cognitiva. Vale referência também ao papel de Freud, ao descrever o processo de introspecção que levou ao inconsciente.

A linguística teve a sua importância a partir dos trabalhos de Skinner (1957) e a posterior formalização da linguística por Chomski. A importância desta contribuição se faz presente num capítulo importante da IA, chamado PNL (processamento em linguagem natural), uma das primeiras iniciativas da IA.

Finalmente, a computação teve papel fundamental na IA, já que para que esta exista, deve-se ter inteligência+artefato (este, o computador).

Estudos recentes apontam que o computador foi inventado em 3 locais/datas distintos, mas mais ou menos ao mesmo tempo. O primeiro é Turing que construiu o Heath Robinson por volta de 1940. Melhoramentos posteriores levaram ao Colossus, construído em 1943. Konrad Zuse, cientista alemão, construiu o computador Z3 na Alemanha em 1941. As inovações deste projeto foram o ponto flutuante e a primeira linguagem de programação conhecida, chamada PlanKalkul. Finalmente, John Atanasof, construiu o ABC nos Estados Unidos, em 1941.

Perceba-se que o John Mauchly, universalmente reconhecido como criador do computador ENIAC, nem aparece nesta lista, já que o ENIAC começa a funcionar em 15 de fevereiro de 1946.

O grande impulso comercial do computador foi o lançamento da família 701 pelo fabricante IBM, no ano de 1952. A partir daqui a IBM se transformou no maior fabricante de computadores da história.

3.2.3 O começo

MmCullogh e Pitts propuseram em 1943 uma Rede Neural Artificial. Foi uma construção intelectual impressionante, resultado de no mínimo 3 *inputs* a saber:

- A fisiologia de um neurônio natural. Conhecida a partir dos trabalhos do médico espanhol Ramon y Cajal, no início do século XX.
- Lógica Proposicional formulada e sistematizada por Russell
- Teoria da Computação, a partir dos trabalhos de Turing

Shannon em 1950 e Turing em 1953 chegaram a escrever programas de computador para jogar xadrez.

Em 1956, organizou-se um seminário de verão, com duração de 2 meses no Dartmouth College, com a finalidade de conectar pesquisadores e de avaliar o estado da arte. Atenderam ao seminário 10 pessoas. Duas foram as conseqüências importantes:

 ©88-09, Pedro Kantek

Na primeira, os pesquisadores se conheceram e tiveram seus trabalhos interfecundados. Na segunda, foi acordado o nome para este ramo novo de conhecimento: **Inteligência Artificial**.

O programa TL (Teórico Lógico) construído por Newell e Simon na Carnegie Tech (hoje Universidade Carnegie Mellon), deu uma demonstração de um dos teoremas do Principia Mathematica (de Bertrand Russel) que era mais curta que a original do autor. Este fato entretanto não sensibilizou os editores do *Journal of Symbolic Logic* que recusaram um artigo de Newell, Simon e TL descrevendo o fato.

No começo hove grandes progressos, entretanto já nessa época os céticos diziam que o computador jamais poderia fazer X. Os pesquisadores respondiam a isso implementando X. Essa época é conhecida como *olhe mamãe, agora sem as mãos.*

> A lista de X original de Turing é: ser amável, polifacético, formoso, amigável, ter iniciativa, senso de humor, saber distinguir o certo do errado, cometer erros, apaixonar-se, gostar de morangos com creme de leite, fazer com que alguém se apaixone de uma pessoa, aprender com a experiência, usar as palavras corretamente, ser objeto de seus próprios pensamentos, ter uma conduta diversa da do homem e fazer coisas realmente novas

O TL deu origem ao GSP, que implementou o pensar como um humano. Em 1952, Arthur Samuel criou um jogador de damas, que rapidamente passou a ganhar de seu criador. Este fato deitou por terra a alegação de computadores só fazem o que lhes dizemos. Apresentado na TV em fevereiro de 1956, o programa causou funda impressão. Samuel era engenheiro da IBM e usava computadores dentro da fábrica, que ainda não haviam sido entregues.

John McCarthy (que já fora o autor do nome IA) fêz 3 colaborações de peso:

- a construção da linguagem LISP (LISt Processor), verdadeiro paradigma na programação de IA até os nossos dias. É a segunda linguagem de programação mais antiga ainda em uso, só perdendo para o FORTRAN.
- a tecnologia de tempo compartilhado. Essa idéia acabou implementada e os responsáveis construiram a Digital Co, durante muito tempo a segunda maior fabricante de computadores, e que implementava em toda a linha o tempo compartilhado.
- A escrita do artigo *Programs with common sense*. Embora com quase 50 anos este artigo continua atual. Ele propõe que programas usem raciocínio mais algum conhecimento do mundo, e esta segunda parcela foi a novidade apresentada.

Em 1958, Marvin Minsky foi para o MIT e junto com McCarthy começou a definir o campo de estudo. A partir daqui surgiram os micromundos:

- Slagle, 1963, SAINT, que fazia cálculo de integrais
- Evans, 1968, ANALOGY, respondia testes de QI
- Bobrow, 1967, STUDENT, resolvia problemas de álgebra do colegial
- Winograd, 1972, programa que compreendia linguagem natural
- Huffman, 1971, programa que manipulava blocos sobre uma mesa
- Waltz, 1975, programa que olhava e entendia os blocos de Huffman
- Rosenblat, 1962, Perceptron, programa que aprendia (RNA)

- Weizenbaum, 1965, ELIZA, uma brincadeira que foi levada a sério.

Nos anos 70, viu-se que tinha havido otimismo de sobra. De fato, a primeira aplicação "comercial" de IA era um tradutor russo-inglês, motivado pelo receio dos EUA de perderem a guerra (fria) com a URSS, na esteira do lançamento do Sputnik em 1957. Este programa fora um fracasso, traduzindo por exemplo, "O espírito é forte, a carne é fraca" por "a vodka é boa, mas a carne está podre".

Um relatório do governo americano em 1966 afirmou que nenhuma tradução havia sido gerada e com isso contratos e subvenções foram cancelados.

Antes da teoria NP (devida a Cook, 1971) se achava que para resolver problemas reais bastava agregar mais hardware e mais memória. Em 1973, através do relatório Lighthill, o Governo Britânico retirou a ajuda generalizada na pesquisa de IA mantendo-a apenas em 2 universidades. Este relatório citou a incapacidade dos programas em lidar com a explosão combinatória.

3.2.4 Futuro

Ninguém consegue exatamente predizer o futuro da IA. A discussão parece se dar entre 2 crenças: a chamada **IA forte** afirma que as máquinas terão consciência em algum momento no futuro da humanidade. Certamente este momento não está no nosso horizonte de tempo, mas ainda assim, cientistas que militam aqui (possivelmente a minoria deles) afirmam que um dia o computador terá consciência (e suas conseqüências: personalidade, dor, preguiça, inveja, amor...). Já a chamada **IA fraca** sugere que a falta de consciência, não é devida a insuficiência de recursos de hardware ou software, mas que é uma impossibilidade. Não importa o que se invente e o que se produza, nunca as máquinas terão as habilidades acima.

Encerra-se esta digressão afirmando que esta discussão remonta a Platão e Aristóteles, quando estes perguntavam se temos alma, já que é aparentemente disso de que falam os cientistas quando discutem IA forte ou IA fraca. Uma boa discussão deste tema está no livro "A nova mente do Rei", de Roger Penrose.

3.2.5 Exercício

Resuma a aula de hoje conceituando ou comentando:

1. O teste de Turing
2. O teste total de Turing
3. existe IA sem computador ?
4. existe IA sem seres humanos ?
5. IA faz parte da ciência cognitiva ou da ciência da computação ?
6. dê um exemplo prático da maldição da dimensionalidade
7. dê um exemplo prático de um *toy domain*

Dividam-se em grupos de 4 ou 5 alunos e realizem o exercício coletivo, conforme as regras que estão lá determinadas.

 ©88-09, Pedro Kantek

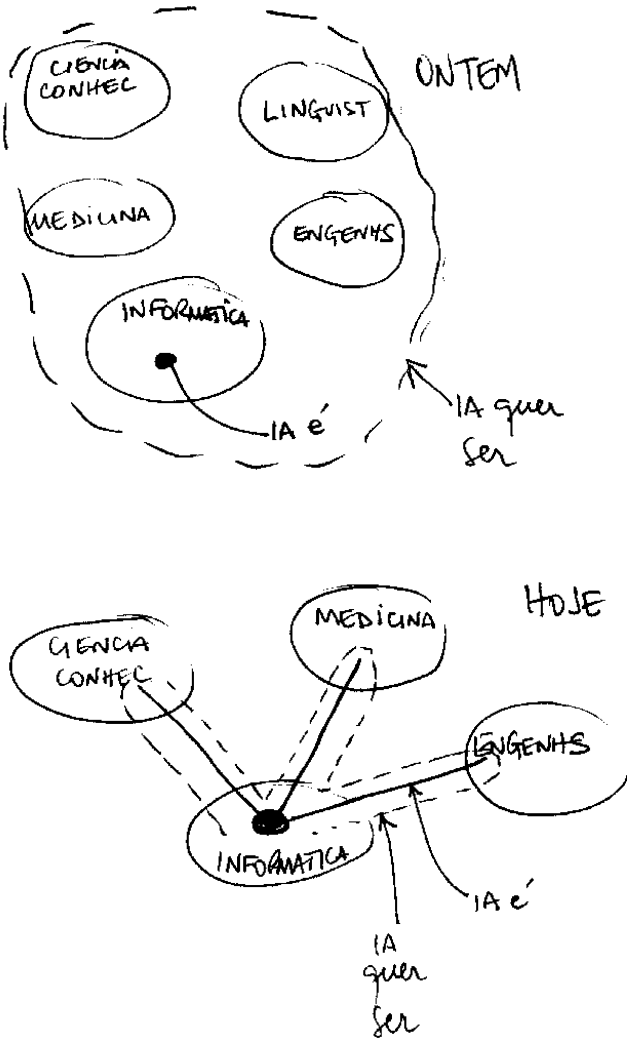


Figura 3.1: Uma visão pessoal da IA, ontem e hoje

Desafio

- Suponha que você é o interrogador no Teste de Turing. Escreva 3 perguntas "bacanas"com a finalidade de desmascarar o programa.
- Suponha que você é o paradigma humano do Teste de Turing. Escreva as respostas para as 3 perguntas acima.
- Suponha que você é o programador do paradigma computador do Teste de Turing. Que habilidades seriam necessárias ao programa para sair-se bem das 3 perguntas acima ?

Crônica de LFV

Sugere-se a leitura da crônica de LFV, denominada Empáfia. A sugestão é que façamos uma rodada de comentários sobre o fenômeno apontado pelo autor.

Empáfia

O Jorge Furtado comprou um programa de traduções para o seu computador e fez uma experiência. Digitou toda a letra do nosso Hino Nacional em português e pediu para o computador traduzi-la sucessivamente em inglês, francês, alemão, holandês, etc. Do português para o inglês, do inglês para o francês e assim por diante até ser traduzida da última língua do programa de volta para o português. Segundo o Jorge, a única palavra que fez todo o circuito e voltou intacta foi "fúlgidos". Em inglês, "salve, salve"ficou "hurray, really hurray"e parece que em alemão o texto ficou irreconhecível como hino mas, em compensação, reformulou todo o conceito kantiano do ser enquanto categoria transcendental imanente em si.

Na época pensei em sugerir ao Jorge que fizesse outro teste e pedisse para o computador traduzir um texto em que constasse a expressão "barato estranho", só para darmos boas risadas. Confesso que o meu barato é ver computador ridicularizado. Uma implicância mesquinha, reconheço. Veja-a como um último gesto de resistência à beira da obsolescência. Não posso mais viver sem o computador, mas a antipatia cresce com o convívio. O programa de texto que uso agora é à prova de erro ortográfico. O computador não me deixa mais errar, por mais que eu tente. Subverte o que eu tenho de mais pessoal e enternecedor e sublinha a palavra errada em vermelho insolente.

Seus critérios nem sempre são claros. A palavra "agora", aí em cima, por exemplo, apareceu na tela sublinhada. Ele está provavelmente sugerindo que talvez eu quisesse escrever "ágora", praça das antigas cidades gregas . Não:

"ágora"também saiu sublinhada. Sua mensagem é que eu tenho uma escolha entre as duas palavras, sua insinuação é que eu não sei a diferença. Não raro repito o erro várias vezes, para mostrar que alguns dos nossos ainda não se intimidaram e na esperança de que ele desconfie de que eu esteja certo - ou seja um caso perdido - e retire a correção. Ainda não aconteceu.

Computador não desconfia. Computador não tem dúvidas. Computador não tem senso de humor. Minha vingança é que, com toda a sua empáfia, ele é um fracasso como tradutor. (Eu sei que "fracasso"não é com cê cedilha, estava só testando).

3.3 Sistemas Baseados em Conhecimento

Parte do fracasso na década de 70 dos sistemas de IA, se deveu ao uso de softwares e de estruturas de dados absolutamente gerais. Para alcançar sucesso, maior especificidade era requerida. A resposta a isso foi a redução do escopo dos sistemas. Agora os sistemas começaram a ser construídos em domínios restritos.

3.4 Agentes

Uma visão bem moderna da IA apresenta como escopo para a IA a construção de agentes inteligentes racionais. Um agente é uma entidade que percebe coisas através de sensores e atua sobre o ambiente através de efetores. A analogia com um agente humano é quase instantânea: o homem tem olhos, ouvidos, pele, nariz (sensores) e mãos, pernas, boca, ... como efetores. Uma característica importantes dos agentes é que eles devem saber coisas.

Um agente racional deve fazer sempre a coisa certa. Embora essa discussão seja ampla, pode-se dizer que a coisa certa é aquela que faz o agente ter sucesso. Isso nos deixa o problema de decidir quando e onde avaliar o sucesso do agente.

 ©88-09, Pedro Kantek

A medida de racionalidade de um agente depende de 4 coisas:

- A medida de desempenho que define o grau de sucesso.
- Tudo o que o agente tenha percebido até agora. Chama-se a história completa das percepções de "seqüência de percepção".
- O que o agente sabe sobre o ambiente
- As ações que o agente pode executar

Com isto temos a definição de um agente racional ideal: Para cada seqüência de percepções possível, um agente racional ideal deveria executar qualquer ação esperada para maximizar a medida de desempenho, baseada nas evidências providas pela seqüência de percepção e por qualquer conhecimento prévio que o agente tenha.

3.4.1 Mapeamento entre seqüência de percepção e ações

Uma possibilidade teórica na construção de agentes seria mapear todas as seqüências de percepção e a ação otimizada que o agente deveria tomar em cada uma delas.

Para a maioria dos agentes que se possa imaginar, esta lista seria muito grande, ou até infinita. Uma maneira de diminuir esta lista é considerar um limite no tamanho das seqüências estudadas na lista. Esta lista, é chamada mapeamento da seqüência de percepções para a ação. A lista não precisa ser construída em sua totalidade, podendo ser substituída por uma lei de formação (quando esta estiver disponível, naturalmente). Tal procedimento, quando possível, economiza tempo e espaço.

Um exemplo em que esta lista poderia ser totalmente construída é no jogo da velha, na qual pela pequena dimensão do tabuleiro, permite um mapeamento completo das ações que um agente deveria tomar para não ser derrotado.

Um exemplo onde tal construção é obviamente impossível é o xadrez. Se pensarmos em um fator de ramificação de 40 (isto é, a cada jogada de um dos contendores, o outro poderá responder com 40 lances diferentes – em média), e uma partida com 50 lances de cada lado, dará origem a uma árvore que SE pudesse ser construída teria 40^{100} nodos. [2] Trazendo essa base para a decimal, e lembrando que $10^{1.60206} = 40$ tem-se que a árvore terá 10^{160} nodos. Apenas como comparação, o número total de eletrons do universo é estimado em 10^{80}.

3.4.2 Agente como um programa

A IA lida com a construção de agentes na forma de programas de computador. Estes deverão rodar em algum substrato, o que será chamado de arquitetura do agente. A arquitetura poderá ser apenas um computador, ou um computador e diversos componentes especiais: câmeras, sensores, rodas, filtros. Em geral diz-se que a arquitetura atua para disponibilizar os dados dos sensores ao programa, roda o programa e transfere para os efetores as ordens dadas por este.

Uma maneira interessante de representar o conjunta foi proposta por Norvig e Russel e denominada descrição PAGE, de perceptores, ações, objetivos (goals) e ambiente (environment). Veja-se na tabela **??** uma tabela extraída do livro deles, à pág. 37.

[2] Lembrar que neste tipo de construção a base é o fator de ramificação da árvore e o expoente é a altura da árvore.

Um exemplo do diagrama PAGE. Veja-se a seguir um diagrama PAGE já preenchido.

Tipo de agente	Perceptores	Ações	Objetivos	Ambiente
Sistema de diagnose médica	sintomas, respostas do paciente	perguntas a fazer, testes clínicos, tratamento	saúde do paciente, minimização de custos	paciente, hospital
Sistema de análise de imagens de satélite	pixels com cor e intensidade variáveis	cena categorizada	categorização correta	imagens de um satélite orbital
Robot selecionador de peças	pixels de intensidade variável	pegar peças e jogá-las no container correto	partes colocadas no container certo	esteira rolante e peças
Controlador de uma refinaria	leituras de temperatura, pressão, fluxo	abrir e fechar válvulas, controlar temperatura	Maximizar a pureza, lucro, segurança	refinaria
Tutor de inglês interativo	palavras digitadas	imprimir exercícios, sugestões e correções	maximizar a nota dos alunos	conjunto de estudantes

Poder-se-ia construir um PAGE para o teste de Turing. Eis como ficaria, na tabela a seguir.

PAGE do teste de Turing. Diagrama PAGE para o Teste de Turing.

Tipo de agente	Perceptores	Ações	Objetivos	Ambiente
Um aspirante a passar no teste de Turing	Perguntas do questionador	Respostas ao questionador	ser confundido com um ser humano	conhecimento do universo

Exercício Preencha o diagrama PAGE para os seguintes agentes

 ©88-09, Pedro Kantek

tipo do agente	percepção	ação	meta	ambiente
controlador de uma câmera fotográfica digital				
motorista de taxi				
browser inteligente				
tradutor português espanhol				

3.4.3 Algoritmo básico de um agente

O algoritmo básico de um agente poderia ser estabelecido como descrito no algoritmo abaixo.

Entrada: PERCEPÇÃO
Saída: AÇÃO

1: **função** Agente
2: static MEMÓRIA
3: MEMÓRIA ← ATUALIZA-MEMÓRIA(MEMÓRIA, PERCEPÇÃO)
4: AÇÃO ← ESCOLHA-MELHOR-AÇÃO(MEMÓRIA)
5: MEMÓRIA ← ATUALIZA-MEMÓRIA(MEMÓRIA, AÇÃO)
6: devolva AÇÃO

Note-se que não faz parte do algoritmo básico o julgamento da medida de desempenho do agente. Usualmente é uma entidade externa que faz este julgamento.

Na construção de agentes, 4 abordagens podem ser escolhidas, usualmente mescladas. São elas:

- Agentes baseados em reflexo simples. Este tipo de agente é caracterizado por uma busca local.
- Agentes que "lembram"aspectos do universo. Ao contrário, este considera uma busca global.
- Agentes baseados no atingimento de objetivos. Considera o aspecto de eficácia.
- Agentes baseados na medida de utilidade. Considera o aspecto de eficiência.

3.5 Agentes inteligentes

Um agente percebe seu ambiente através de sensores, raciocina e decide internamente e depois age sobre o mesmo ambiente através de atuadores ou efetores. Vejam-se alguns exemplos:

agente	sensores	atuadores
homem	olhos, ouvidos	mãos, boca, pernas
robot	câmeras, telêmetros	motores
software	strings binários	string binários
cachorro		
gato		
multador da BR		

Um agente racional é aquele que obtém o **melhor desempenho**. A questão agora é como e quando medir. Uma hipótese simples é perguntar ao agente. Este pode responder de maneira adequada ou não, talvez até não saiba como responder. Agentes humanos, podem ser tentados a responder *as uvas estavam verdes* [3] quando perguntado porque não atingiu seu desempenho esperado. É melhor contar com uma autoridade externa ao agente que seja capaz de medir. [4] Esse mesmo observador externo deve ser capaz de estabelecer a norma do que é satisfatório em termos de desempenho. Nunca é demais reforçar o cuidado com o estabelecimento de metas. Por exemplo, suponha um aspirador de pó autônomo. Se a sua meta de desempenho for recolher 5Kg de pó dentro da área de limpeza, ele pode ser tentado a derramar 10Kg de sujeira e depois recolher os 5Kg esperados. [5]

Fatores de racionalidade

- Medição de desempenho
- Seqüência de percepções (o que o agente percebe do universo)
- Conhecimento do agente sobre o meio
- Ações possíveis ao agente

Seja um exemplo hipotético sobre um agente policial reprimindo o tráfico de drogas, e considerando os fatores de racionalidade de trás para a frente. As ações possíveis ao agente estão claramente delimitadas na legislação sobre o assunto. O conhecimento do agente sobre o meio pode ser representado pelo treinamento e pela experiência do referido agente. A seqüência de percepções é o trabalho do dia a dia do agente e tem a ver com suas habilidades profissionais. Finalmente, a medição do desempenho pode ser, por exemplo, a quantidade de droga apreendida.

Agente racional ideal

Um agente racional ideal deve empreender ações que maximizem seu desempenho, baseando-se na seqüência de percepções e no conhecimento que esteja incorporado nesse agente.

Questão para discussão Um relógio pode ser considerado um agente ?

[3] Lembrando a fábula de La Fontaine – a raposa e as uvas. Ia uma raposa pelo campo, com fome, quando viu uma parreira carregada de uvas. Imediatamente desviou-se do caminho pensando que estava resolvido o problema da fome. Chegando ao pé da árvore, viu que as uvas estavam muito altas. Ainda assim, tentou por muito tempo, através de saltos, atirando pedras, procurando um bambu para cutucar a árvore. Horas mais tarde, cansada e percebendo que não consiguiria recolher nenhuma uva, desistiu, dizendo a si própria *Não faz mal, as uvas estavam verdes.*

[4] a respeito desta questão, verifique-se a enorme rejeição do poder judiciário em nosso país à simples hipótese de uma avaliação externa

[5] Uma vez vi uma biblioteca cuja meta de desempenho para o ano era agregar 5m de livros nas estantes

 ©88-09, Pedro Kantek

3.5.1 Propriedades do ambiente

Ambientes tem diversas categorias. Eis algumas oposições possíveis, que ajudam a clarear as idéias:

Acessível versus inacessível Se os sensores do agentes têm condições de acessar a integridade das características importantes do ambiente para a escolha da melhor ação diz-se que o mesmo é acessível. Esta característica é importante porque desobriga o agente a "lembrar"aspectos ou características prévias.

Determinístico versus não determinístico Um ambiente é determinístico quando o próximo estado do mesmo é perfeitamente determinado a partir das condições atuais. Nenhuma randomicidade interfere. Às vezes, o ambiente é inacessível, mas ele **parece** ser não determinístico. É claro que este tema pode dar origem a discussões filosóficas: o universo é determinístico ou não ? Talvez o caos deva fazer sua aparição aqui.

Episódico versus não episódico O agente entende a sua atuação como sendo formada por episódios. Cada episódio é composto pela percepção do agente seguida de uma ação. A qualidade da ação depende apenas da percepção deste episódio, não dos episódios passados. Ambientes episódicos são mais simples, o agente não precisa pensar à frente.

Estático versus dinâmico Se o ambiente pode trocar enquanto o agente está deliberando, diz-se que o ambiente é dinâmico para o agente. Em ambientes estáticos o agente não precisa preocupar-se com a passagem do tempo. Se o mundo não muda, mas o agente precisa preocupar-se com o limite de tempo, diz-se que o ambiente é semidinâmico.

Discreto versus contínuo Se há um número limitado de percepções distintas, claramente definidas, diz-se que o ambiente é discreto. Ao contrário será contínuo quando houvem uma gama ilimitada de opções a serem feitas pelo agente.

Claramente o pior caso (do ponto de vista de quem tem que construir o agente) é o ambiente inacessível, não-episódico, dinâmico e contínuo.

Seja um quadro no qual alguns ambientes de sistemas IA estão listados com suas características. Note que as opções feitas na tabela **??** podem ser questionadas a partir de como são conceituados os ambientes e os agentes.

Possíveis escolhas em ambientes e agentes na tabela a seguir, algumas análises de características de agentes

ambiente	acessível	determinístico	episódico	estático	discreto
pingue (xadrez rápido)	sim	sim	não	semi	sim
xadrez	sim	sim	não	sim	sim
poquer	não	não	não	sim	sim
dirigir um taxi	não	não	não	não	não
diagnóstico médico	não	não	não	não	não
análise de imagem	sim	sim	não	semi	não
robô classificador	não	não	sim	não	não
controlador de refinaria	não	não	não	não	não
auxiliar de redação em inglês	não	não	não	não	sim
câmera digital					
browser inteligente					
tradutor português-espanhol					

3.5.2 O mundo dos aspiradores de pó

Seja um ambiente que deve ser limpo através de aspiradores de pó inteligentes. Seu mundo pode ser assim descrito:

perceptores Cada agente aspirante recebe um vetor de 3 números

sensor de toque é 1 se o aspirador bateu em algo, 0 senão.

fotosensor de limpeza é 1 se o local está empoeirado e 0 senão.

infrasensor de repouso é 1 se o aspirador está no seu local de armazenamento. 0 senão.

ações São 5 ações: em frente, virar à direita, virar à esquerda, dar um pulso de aspiração e parar.

objetivos O objetivo de cada aspirador é limpar tudo e voltar para o repouso. Uma medida de desempenho poderia ser:

Mais 500 pontos casa totalmente limpa.

Menos 1 ponto para cada ação tomada pelo aspirador

Menos 1000 pontos se ele não estiver em sua posição de repouso quando desligado.

ambiente Uma grade de $n \times n$ células quadradas. Cada célula pode conter um obstáculo (móveis ou paredes) ou ser aberta. Apenas algumas células abertas contém sujeira. Cada ação de "em frente"move o aspirador uma célula à frente na última direção em que ele estava, a menos que esta esteja impedida, quando então o aspirador não se move (e é avisado pelo sensor correspondente).Um pulso de aspiração limpa a célula atual. Parar encerra a simulação.

 ©88-09, Pedro Kantek

Uma proposta de solução Seja uma grade $m \times n$ retangular, contendo os seguintes valores:

1 obstáculo nesta célula

0 célula livre limpa

2 célula livre suja

3 local de armazenamento de um aspirador

Por exemplo, uma possível instância poderia ser

1	1	1	1	1	1	1	1
1	3	0	0	2	0	0	1
1	0	0	2	0	0	2	1
1	1	1	1	1	1	2	1
1	2	0	0	2	0	0	1
1	2	0	2	0	0	2	1
1	1	1	1	1	1	1	1

O algoritmo do ambiente (global) poderia ser

```
1: algoritmo Ambiente
2: CASA ← inicialização {haverá k aspiradores}
3: inteiro CUSTOS [k]
4: CUSTOS ← 0
5: global lógico LIGADO [k]
6: LIGADO [k]← VERDADEIRO
7: inteiro SENSOR [k] [3] {sensores de cada aspirador}
8: ONDE [k] [2] {onde está cada aspirador}
9: ULTIMO [k] {qual a última direção do aspirador k}
10: enquanto houver aspirador LIGADO faça
11:    para j de 1 até k faça
12:       CUSTO[j] ← CUSTO[j] + AÇÃO-ASPIRADOR(j)
13:    fimpara
14: fimenquanto
15: para j de 1 até k faça
16:    se casa limpa então
17:       CUSTO[j] ← CUSTO[j] - (500 ÷ k) {note o sinal de menos}
18:    fimse
19: fimpara
```

Saída: devolve uma parcela do custo de limpeza

```
1: função AÇÃO-ASPIRADOR (inteiro x)
2: inteiro CUSTOLOCAL ← 0
3: se SENSOR [x] [3]=1 então
4:    inicializa CAMINHOx e reseta SENSOR [x] [3]
5: senão
6:    se sensor de toque = 1 então
7:       x VIRAR A DIREITA
8:       CAMINHO x ← CAMINHO x, concatenado com DIREITA
9:       SENSOR[x][1]←0
10:   senão
11:      se fotosensor = 1 então
12:         PULSO ASPIRATORIO x
13:         CUSTOLOCAL ← 1
```

```
14:       senão
15:         ANDAR x {o aspirador x anda 1 casa na direção ULTIMO[x]}
16:         CAMINHO x ← CAMINHO x, concatenado com EMFRENTE
17:         CUSTOLOCAL ← 1
18:       fimse
19:     fimse
20:     se não há mais sujeira na casa então
21:       faça o caminho CAMINHOx em ordem inversa
22:       LIGADO[x] ← 0
23:       devolva o tamanho do vetor CAMINHOx
24:     senão
25:       devolva 1
26:     fimse
27: fimse
```

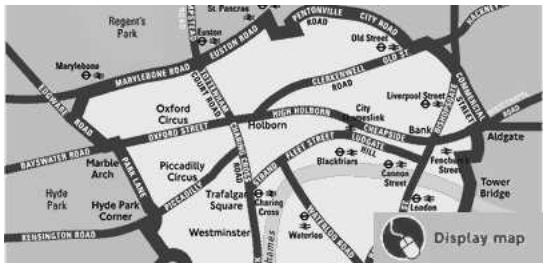
```
1: função ANDA x
2: se a casa para onde vai é 1 então
3:   SENSOR[x][1]← 1
4: senão
5:   marca célula atual da casa com 1
6:   anda na direção indicada
7:   se célula atual = 2  então
8:     SENSOR[x][2]←1
9:   fimse
10: fimse
```

Note que este conjunto de algoritmos não penaliza em 1000 pontos pois ele sempre retorna ao local de origem.

3.5.3 Exercícios

Discuta o diagrama PAGE para os seguintes sistemas que usam IA.

Pedágio em Londres Um sistema que é composto de cerca de 800 câmeras de vídeo que fotografam o trânsito londrino. O perímetro do sistema está mostrado na figura **??**. O sistema começõu a funcionar em fevereiro de 2003. O motorista deve pagar 5 libras por dia para entrar no perímetro da cidade. O pedágio vigora de 7h00 as 18h30 de segunda a sexta. O pagamento pode ser feito por celular, web, telefone e pessoalmente. Na segunda semana de operação, cerca de 500.000 pedágios foram pagos. 30.000 multas por não pagamento foram emitidas (cerca de 40 libras cada) e detectou-se um tráfego 20% menor. Os detalhes podem ser vistos em www.cclondon.com.

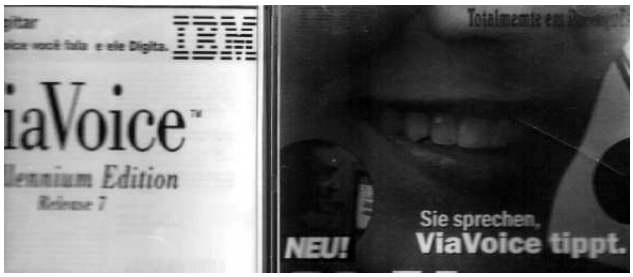


Perímetro do Congestion Charge em Londres

Via Voice Um sistema originalmente produzido pela IBM para captura de voz e transformação desta em texto a ser capturado pelo computador. Serve para "ditar para

 ©88-09, Pedro Kantek

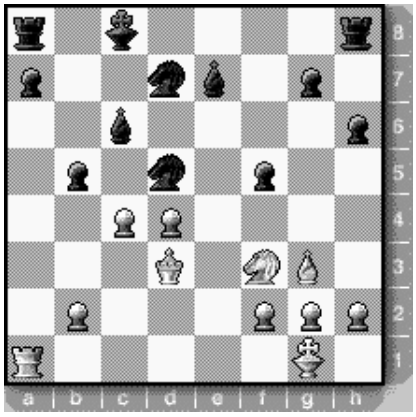
o micro". Funciona razoavelmente bem. O micro precisa ser treinado na voz do dono (Há que se ler Machado de Assis para o deleite do micro...). Veja em **??** a capa do software.



Capa do CD do Viavoice

MYCIN Um dos primeiros sistemas operacionais de IA. Usa uma base de conhecimento sobre doenças infecciosas. Sugere tratamento para os pacientes. Em uma simulação entre um grupo de 9 médicos humanos e o MYCIN, avaliando casos não triviais, os 10 foram convidados a dar sugestões de tratamento. As indicações foram entregues a um comite de infectologistas que deram notas aos tratamentos, na modalidade duplo-cego. O sistema teve o melhor desempenho entre todos. [Yu, V.L. et alli. Antimicrobial Selection for Meningites by a Computerized Consultant - a blinded evaluation by infectious disease experts. JAMA vol 242, 1979, pag. 1279].

Deep Blue Um programa de computador associado a um hardware especialmente preparado pela IBM que conseguiu a proeza de vencer ao campeão mundial de xadrês Gasparov em 1997. Na figura **??** pode-se ver o resultado da última partida, a sexta, que foi vencida por Deep Blue. Deep joga com as brancas e Kasparov abandona no 19. lance. O resultado do match foi 3,5 a 2,5 para Deep Blue. Mais informações em www.researchibm.com/deepblue.



A última partida Deep Blue X Kasparov

O mundo do aspirador Uma proposta de construção de um sistema de aspiradores autônomos para limpar uma casa. Veja a descrição no texto.

Construa a tabela PAGE e o diagrama de características do universo para cada um dos sistemas.

Tipo de agente	Perceptores	Ações	Objetivos	Ambiente
cclondon				
via voice				
MYCIN				
Deep Blue				
Mundo do aspirador				

Ambiente	Acessível	Determinístico	Episódico	Estático	Discreto
cclondon					
via voice					
MYCIN					
Deep Blue					
Mundo do aspirador					

 ©88-09, Pedro Kantek

Capítulo 4

Busca em espaço de estados

4.1 Introdução

Nesta aula vai se estudar um comportamento possível para um agente que precise escolher qual ação tomar através da análise dos resultados obtidos em cada uma das seqüências de ações permitidas.

O nome deste tipo de agente é *agente para a solução de problemas*. Lembrando, agentes úteis devem obter um máximo no seu rendimento. Esta característica é mais facilmente obtida quando o agente tem uma meta. Esta meta deve ser um subconjunto de possíveis estados do mundo que interessa ao agente. As ações possíveis ao agente são as aquelas que causam transições entre um e outro estados do mundo. O trabalho do agente é descobrir quais as ações levarão a um estado meta.

O conceito de *formulação do problema* é o processo de decidir quais ações e estados serão considerados, e vem logo depois de definir a meta.

Exemplos

- ir de Ponta Grossa a Florianópolis de carro
- resolver o cubo Rubik
- vencer uma final de rei e torre contra rei no xadrêz

Em qualquer um destes, no momento inicial, o agente se encontra distante da meta e tem diante de si diversas alternativas. O problema de escolher qual das opções aproxima o agente da meta é conhecido como problema de *busca*. A busca pode ser com e sem contingência. Quando não há contingência, a dificuldade da busca em geral é refletida pelo fenômeno da explosão combinatória. Se tivéssemos capacidade computacional infinita, o problema poderia ser resolvido decidindo primeiro e agindo depois. Quando há contingência, fenômenos imprevisíveis (sobre os quais o agente não tem controle) intereferem no processo de busca, fazendo com que decisões precisem ser intercaladas com ações.

No exemplo acima, os problemas de PG-Floripa e de Rubik são sem contingência e o de final de torre e rei contra rei conta com uma contingência: o adversário. Note, que dependendo da abrangência do problema e do grau de profundidade pretendido para a solução, o desconhecimento do mundo pode ser entendido como contingência também.

4.1.1 Definição de problema

Neste contexto, um problema de busca em espaço de estados é definido pelas seguintes entidades:

estado inicial A maneira como o problema começa

operadores Lista de todos as possíveis ações que o agente poderá tomar na solução do problema

teste da meta Condição a que deverão se submeter os estados. Em caso de resultado verdadeiro, este estado é uma *meta* e o problema está resolvido. Em caso de falsidade, o problema segue em aberto

custo A função de custo assinala um valor a cada operador aplicado. Para certa classe de problemas, espera-se encontrar a solução de menor custo possível.

Os conceitos de estado inicial e operadores definem o que se costuma chamar de *espaço de estados*. Trata-se do conjunto (possivelmente infinito) de estados que se podem alcançar a partir do estado inicial pela aplicação sucessiva dos operadores. Encontrado um estado meta, a assim chamada *solução* do problema é o conjunto de operadores, aplicados em ordem reversa à que foram descobertos, desde o estado inicial até o estado meta.

A verdadeira arte da solução de problemas de busca de estados é saber decidir o que vai servir para descrever estados e operadores e o que não vai servir. Para problemas de cunho real, uma necessária abstração de detalhes terá que ser implementada, a fim de que o problema possa ser tratado. Costuma-se dividir este campo em problemas de quebra-cabeça e jogos versus problemas do mundo real. Os primeiro são adequados para estudar e medir abordagens e algoritmos. Essa é a razão do sucesso do estudo do xadrêz na IA.

Exemplos

Quebra cabeça 15 Sam Loyd (1841-1911) foi um charadista americano que ganhou a vida construindo quebra-cabeças. Ele era contratado por empresários para construir charadas que acompanhavam a publicidade das empresas, nas quais eram oferecidos prêmios para os que resolvessem os problemas. O enigma 14-15 foi o mais famoso de Loyd (segundo Simon Singh, foi o equivalente vitoriano ao cubo Rubik). Ele era vendido na disposição da figura, e era oferecido um prêmio de US$ 1000 a quem o conseguisse resolver. Como se vê o desafio era inverter as peças 14 e 15 (daí o nome do puzzle). Loyd estava confiante de que nunca teria que pagar o prêmio, pois ele havia descoberto um *invariante*. Invariantes são importantes na matemática e na programação, pois eles ajudam a validar estados intermediários de um problema. Para explicar o invariante de Loyd, comece-se com o quebra cabeça resolvido. Aqui, esticando o tabuleiro na forma de um vetor, percebe-se que não existe nenhum par invertido. Definindo um parâmetro como PARES-INVERTIDOS, pode-se afirmar que $PI = 0$ para o problema resolvido. Para qualquer movimento válido que se realize sobre este tabuleiro, verifica-se que o PI sempre é um número *par*.

> **Exercício**: Calcule o PI para diversas configurações possíveis

Este foi o invariante que Sam Loyd descobriu. A partir daqui foi fácil oferecer o prêmio. Ele sabia que o problema era impossível.

estados A descrição de um estado é a localização de cada uma das 15 peças em tabuleiro de 4×4. Por razões de praticidade é adequado representar também o buraco como uma 16^a peça. Os estados inicial e final são dados.

operadores Cada uma das peças adjacentes ao buraco pode se mover em 4 direções, pelo que poderíamos ter 16 operadores. Entretanto, o problema fica simplificado se se fizer uma abstração-simplificação: agora quem se move é o buraco e não a peça. Com essa esperteza, o número de operadores cai para apenas 4.

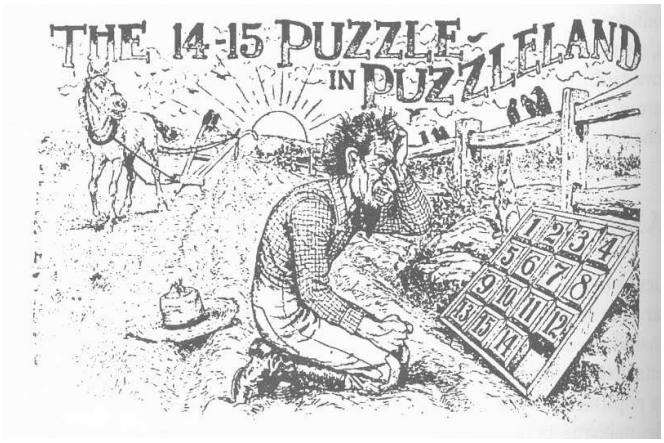


Figura 4.1: O quebra-cabeça de Sam Loyd

condição de fim O tabuleiro deve coincidir com o estado final proposto

custo Se se considerar cada movimento com um custo igual a 1, a função de custo corresponde à quantidade de operadores utilizados. Este problema surgiu por volta de 1870 quando o americano Sam Loyd ofereceu uma grande quantidade de dinheiro a quem resolvesse o quebra-cabeça. Ele sabia que não corria risco ao oferecer o dinheiro, mas aparentemente ninguém mais o sabia.

As 8 rainhas O objetivo deste problema (que pode ser facilmente generalizado para n rainhas) é dispô-las em um tabuleiro de xadrêz de tal maneira que nenhuma rainha ameace as outras já dispostas no tabuleiro. Lembrar que no xadrêz uma rainha pode atacar qualquer peça que esteja na mesma coluna, linha ou diagonal. Embora existam algoritmos exaustivos para a solução deste problema, pelo menos para tabuleiros pequenos (vide WIRTH, Niklaus, Algoritmos e Estruturas de Dados), esta é uma instância interessante de problema para estudo. Existem 2 possibilidades de formulação: na primeira, dita *incremental*, as rainhas vão sendo postas uma a uma no tabuleiro. Na segunda, dita *completa*, as rainhas são colocadas todas de uma vez e daí vão sendo modificadas até a solução final. Neste problema não há relevância para a função de custo. Eis como ele poderia ser descrito

meta As 8 rainhas dispostas, sem que se ataquem

custo Pode-se considerar zero

Quanto aos estados e operadores, existem diversas abordagens. Seja por exemplo

estados Qualquer disposição que tenha de 0 a 8 rainhas no tabuleiro

operadores Pôr uma rainha em qualquer casa

Nesta formulação há 64^8 seqüências para serem investigadas. Usando uma dica do problema na formulação, vai-se considerar apenas casas não atacadas como candidatas a receberem rainhas. Fica assim

estados 0 a 8 rainhas dispostas de maneira que nenhuma ataque a outra.

operadores Colocar uma rainha na coluna vazia (começando pela esquerda)de modo que não ataque nenhuma rainha já colocada.

 ©88-09, Pedro Kantek

Esta estratégia de operadores é mais adequada, embora haja um problema: as vezes nenhuma ação será permitida (fenômeno chamada de *beco sem saida*), pelo que haverá necessidade de retroceder e tentar outra alternativa. Para esta abordagem existem apenas 2057 possíveis alternativas. Perceba-se neste exemplo, que a formulação do problema pode ter conseqüência direta sobre o tamanho do espaço de busca. Este problema surgiu em 1848 quando foi publicado na revista Schach, uma publicação alemã de xadrêz. Republicado em 1850 atraiu a atenção de K.F.Gauss, que buscou localizar todas as respostas possíveis. Achou 75 (são 92).

criptoaritmética São problemas representando operações aritméticas envolvendo palavras. Cada letra terá que ser substituída por um dígito e a operação deve dar certo. Alguns exemplos

```
FORTY              SEND          COCA
+ TEN            + MORE         +COLA
+ TEN             -----         -----
-----             MONEY          SODA
SIXTY
```

cujas soluções podem ser

```
29786
+ 850
+ 850
-----
31486
```

Ache a resposta de SEND + MORE = MONEY Este problema poderia ter a seguinte formulação

estados Cada letra é substituída por um valor distinto

operadores para todas as ocrrências de cada letra, substitua-a por um valor ainda não usado

meta a operação indicada, já com os números substituídos, está correta

custo Não há custo, todas as soluções são igualmente válidas.

Missionários e canibais Três missionários e três canibais se encontram em uma margem de um rio. Precisam atravessá-lo e contam para isso com um pequeno bote que apenas leva 2 pessoas. A restrição do problema é que em nenhum momento um missionário deve estar junto com 2 ou 3 canibais (viraria churrasquinho) e nem um canibal deve estar com 2 ou 3 missionários (ouviria um sermão). Naturalmente, o barco precisa ser conduzido por pelo menos 1 pessoa.[1]. Este problema surgiu em 1961 (Simon e Newell) e foi a primeira vez em que um problema trivial e coloquial foi tratado formalmente para solução em computador. Serve também como paradigma de abstração de detalhes no momento de implementar o solucionador.

Eis uma possível definição do problema

estados seqüência de 3 números, que representam a quantidade de missionários, canibais e barcos que estão na margem onde o problema começa.

[1] Uma versão alternativa pede que em nenhum momento haja mais canibais que missionários. Aqui não há problemas de sermão

operadores A partir de cada um dos estados, os operadores seriam levar 1 missionário, 1 canibal, 2 missionários, 2 canibais ou 1 de cada. No máximo haverá 5 operadores para cada estado, descontando-se aqueles que levam a estados ilegais. Note-se que se cada um dos 6 personagens for individualizado, passam a ser 27 operadores em vez de 5.

meta O estado (0,0,0)

custo O número de vezes que se atravessa o rio.

O tamanho do espaço de estados é muito pequeno para o computador resolver. Também o é para os humanos, mas estes são atrapalhados pelo fato de que alguns operadores parecem "retrógrados". Parece que humanos exigem que suas buscam avancem sempre.

Problemas reais

- Problema de rotas. Por exemplo, seja visitar uma única vez um número qualquer de cidades minimizando o deslocamento. Este é o problema do caixeiro viajante. Este problema é NP, pelo que tem sido muito estudado em busca de soluções heurísticas de menor complexidade.

- Projeto de VLSI. Uma das principais dificuldades na construção de circuitos VLSI é a localização de células e roteamento de conexões. As restrições são minimizar a área das conexões (manter a placa pequena) e o comprimento das mesmas (otimizar a velocidade de operação).

- Robot ambulante. É uma generalização do problema do estabelecimento de rotas. Ao invés de usar uma grade discreta, um robot móvel pode deslocar-se através de um espaço contínuo, em um conjunto praticamente infinito de operadores. Se o robot apenas se desloca, o espaço de busca é bidimensional, se o robot tem braços e pernas que se devem controlar o espaço de busca vira multidiomensional. Para converter o espaço de busca em finito, há que se usar técnicas especializadas.

- Seqüência de montagem de peças complexas. Este problema ganha destaque nas linhas de produção automatizadas cada vez mais freqüentes em manufaturas.

Busca de soluções

Estabelecido o estado inicial e a lista de operadores, lembrando que alguns deles podem conduzir a becos sem saída, chega o momento de efetuar a busca. Para tanto, haverá que escolher 1 operador na lista e gerar o estado subsequente. Agora, no mínimo há dois estados candidatos a gerar sucessores (o original e o que foi obtido no passo anterior). Para cada um há que escolher qual sucessor aplicar, e assim por diante, lembrando sempre que para cada novo estado gerado há que se testar a condição de fim. Há dois cuidados aqui: não repetir operadores já usados em um estado a fim de evitar a introdução de loops na sequencia de solução, e cuidar que nenhum estado gerado seja inválido. Assim, a busca de solução passa pela série de operações escolher, testar meta, expandir
O processo termina quando se encontra a meta ou quando não há mais expansões a realizar, e neste caso o problema não tem solução. Uma boa sugestão é pensar no espaço de estados como uma árvore de busca. O estado inicial é a raiz da árvore, e estados que não possam gerar sucessores (por serem a meta ou por somente gerarem estados inválidos) serão folhas. O nodo que está sendo presentemente expandido é conhecido como nodo de busca.

Os nodos da árvore de busca devem conter no mínimo:

 ©88-09, Pedro Kantek

estado que corresponde ao nodo	o nodo pai deste	o operador que foi usado	profundidade deste nodo	custo da rota desde a raiz até aqui

É preciso guardar na árvore o conjunto de nodos a espera de explosão. São conhecidos como *margem* ou *fronteira*.

Qual o papel do conhecimento ?

Existem problemas que exigem pouco conhecimento e outros que exigem muito. Isso nada tem a ver com a facilidade de obtenção da solução. Por exemplo, um programa de xadrez precisa ter menos conhecimento, ao passo que um programa de entendimento de linguagem natural precisa ter inúmero conhecimento. A Veja-se por exemplo: uma frase em inglês coloquial é dita e depois são feitas afirmações sobre a frase inicial, devendo-se descobrir a quem se refere o pronome "ele"em cada uma das frases (aqui devidamente traduzidas). A frase original é O gato tentou comer o passarinho no porão. A seguir, vem as questões:

- Ele fugiu voando. (Quem é ele ? - o passarinho).
- Ele conseguiu e lambeu gostosamente o focinho (ele é o gato).
- Ele tinha muitas janelas abertas, e possibilitou a fuga (ele é o porão).
- Finalmente, em inglês há mais uma opção por causa da exigência daquele idioma de que o objeto esteja implícito em todas as frases, e que seria: it's too bad !, o que pode ser traduzido por ele (o fato) é muito feio!

Estratégias de controle

Até agora não se disse qual operador ou regra seguir para chegar ao resultado final. Esta questão surge, porque as vezes, mais de um poderá ser aplicado, e as vezes todos poderão ser. A decisão de como operar terá impacto dramático sobre a velocidade e a alcançabilidade da solução.

Quem decidirá que operador aplicar será uma estratégia de controle que deverá ter as seguintes características:

Deve causar movimento. Imagine-se o problema das jarras. Se se aplicar sempre a primeira regra possível (que é: encher a jarra de 4 l), o problema nunca se resolveria. Eternamente se estaria a encher a jarra de 4l.

Deve ser sistemática: No mesmo exemplo, a alternativa da escolha poderia ser a escolha aleatória. Agindo assim, não há garantia de que o problema se resolvesse.

4.1.2 Estratégias de busca

Para estudar as diversas estratégias de busca usam-se 4 critérios, a saber

completude A estratégia garante encontrar solução, se esta existir ?

complexidade temporal Quanto tempo é necessário para encontrar uma solução ?

complexidade espacial Quanta memória é necessária ?

otimalidade Se existirem diversas soluções, a estratégia encontrará a melhor (ou mais genericamente, encontrará uma melhor do que o parâmetro x) ?

Nesta aula serão estudadas as estratégias que não usam informação do problema, também conhecidas como *busca cega*.

São as seguintes as buscas cegas

Busca em largura

Somente se passará a um novo nível na árvore depois que todos os nodos do nivel atual tiverem sido expandidos. Para implementar esta estratégia usa-se uma fila na qual vão sendo colocados todos os nodos expandidos no final da fila. Esta estratégia sempre acha a solução, e se houver mais de uma encontrará a de menor custo (número de arestas entre a meta e a solução). Assim, esta estratégia passa nos critérios de completude e otimalidade. A notícia ruim é que dependendo do problema ela não passa nos outros dois. Suponha-se um problema no qual a expansão de um nodo gera x outros nodos. Diz-se que x é o fator de ramificação da árvore. Olhando para a árvore, ver-se-á que no nivel 1, existem x nodos. Já que cada um deles gera x sucessores, no nível 2 da árvore existem x^2 nodos. No terceiro, x^3 e no n-ésimo, existem x^n nodos. Para um problema cuja solução seja um caminho de comprimento c, haverá que pesquisar $1+x+x^2+x^3+...+x^c$ nodos, no máximo, já que a resposta poderá aparecer em qualquer nodo no c-ésimo nivel.

Aqueles que conhecem a análise de algoritmos ficam nervosos (ou emocionados, depende do tipo de gente) quando aparece uma expressão de complexidade envolvendo exponencial, tal como $O(x^c)$. Veja-se porque:

Seja um problema rodando em um computador que tem fator de ramificação = 10 (nada muito assustador na prática). Vai-se priorizar o estudo sobre a complexidade temporal, já que a espacial pode ser minimizada usando-se memória auxiliar. Supondo que o computador usado consegue gerar 1000 nodos por segundo, e cada nodo ocupa 100 bytes, tem-se

profundidade	qtd nodos	tempo	memória
0	1	1miliseg	100 bytes
2	111	0.1 seg	11 Kbytes
4	11.111	11 seg	1 Mbytes
6	10^6	18 min	111 Mbytes
8	10^8	31 horas	11 Gbytes
10	10^{10}	128 dias	1 Tbyte
12	10^{12}	35 anos	111 Tbytes
14	10^{14}	35 séculos	11 Pbytes

Desta tabela pode-se concluir que a estratégia de busca em largura só é aceitável para instâncias pequenas de problemas.

Busca pelo menor custo

Parecida com a busca em largura, esta estratégia privilegia o nodo que pertencer à margem e que tenha menor custo. O algoritmo pode ser implementado através de um *heap* ao invés de uma fila. Seja por exemplo, a tarefa de ir de K a M no grafo:

	K	A	B	C	M
K		1	5	15	
A					10
B					5
C					5
M					

Neste problema a menor rota é a que leva a um custo de 10.

 ©88-09, Pedro Kantek

Busca em Profundidade

Sempre se expande o nodo que estiver mais distante da raiz. Apenas se a busca é interrompida por um beco sem saída, é que se retorna a outro estado para continuar a busca. O algoritmo é o mesmo da busca em largura com a diferença que se usa uma pilha em lugar de uma fila na geração de candidatos à expansão. A necessidade de espaço é menor do que lá (Para uma ramificação r e uma profundidade p, são necessários armazenar apenas $r \times r$ nodos ao invés de r^p do caso em largura). A desvantagem é que na maioria dos problemas reais a árvore é muito profunda, mesmo infinita as vezes, e neste caso, um mau começo pode levar a um processo muito demorado ou até infinito. É comum implementar a busca em profundidade na forma de uma função recursiva, o que simplifica a pilha de nodos a expandir.

Busca em profundidade com limite

É uma busca em profundidade com um fator de corte. Ao ser alcançado, há um retorno. Esta estratégia encontrará a solução, mas nada garante que seja a melhor solução. Esta busca é completa, mas não ótima. Os requisitos de tempo e espaço são os mesmos da busca em profundidade. A dificuldade é estabelecer de antemão qual seria o limite adequado para cada problema.

Busca com aprofundamente interativo

É a resposta para o estabelecimento de um limite do processo anterior. Ao invés de escolher um limite pré-determinado, este método testa todas as possibilidades. Primeiro a profundidade zero (trivial), depois a profundidade 1, depois a dois e assim por diante. Este método dá a impressão de desperdício, uma vez que nodos não folha são calculados mais de uma vez cada um (quanto menor o nível mais vezes eles são calculados). Por exemplo, supondo este método com fator de ramificação 10 e com aprofundamento interativo de 1 até d níveis, expande 11% nodos a mais que a busca em largura ou em profundidade até o nível d.

Busca Bidirecional

A idéia é começar nas duas direções: do estado inicial em direção à meta e da meta em direção ao início. O processo se encerra quando um mesmo estado é encontrado nas duas árvores. Em tese é um método bom, veja-se este exemplo. Supondo um problema com fator de ramificação f e que tem uma solução na profundidade p, por hipótese $f = 10$ e $p = 6$. A busca em largura produzirá 1.111.111 nodos. Se a busca for nas duas direções, a profunidade agora será de apenas 3 para cada uma das pesquisas. Cada uma gerará 1.111 nodos, e no total serão 2.222 nodos, número muito menor do que o milhão de nodos acima. As dificuldades que surgem são:

- Para esta estratégia há que se definir o *predecessor* além do sucessor. Dependendo do problema não será tarefa fácil, sequer possível.
- Supondo a existência de diversos estados meta, qual escolher para retornar ?
- Para cada estado gerado em uma das árvores, há que se varrer toda a outra para verificar se há casamento.

Exercícios

1. Resolva

```
  donald       cross
+ gerald     + roads
--------     -------
  robert      danger
```

2. Nos antigos teste de QI era comum aparecerem problemas de predição de seqüências. Dados 4 ou 5 números que seguiam uma ordem desconhecida, o leitor era convidado a escrever o próximo. A maneira de resolver o problema era intuir uma lei de formação, testá-la com a seqüência existente e se houvesse sucesso, usar a lei para predizer o próximo número. Por exemplo, se a seqüência fosse 0, 2, 4, 6, o próximo número seria 8. A lei de formação poderia ser $2 \times n$, com n começando em zero e crescendo em 1 unidade. Para o exercício que segue, suponha que n sempre começa em 0. Pede-se a criação de um sistema de busca em espaço de estados que resolva este tipo de problema. O espaço está formado por todas as funções possíveis de construir usando 1 e n, além das funções $+, \times, -, \div$ e potência. Por exemplo, nesta linguagem, o problema 2^n se traduz por $(1 + 1)^n$.

   (a) Escreva a função que testa o estado meta.

   (b) Escreva a função que acha os sucessores de um estado

   (c) Usando o algoritmo, localize o próximo número das seqüências $[1, 2, 3, 4, 5]$, $[1, 2, 4, 8, 16]$ e $[0.5, 2, 4.5, 8]$

3. Exercício adaptado de Luger, "Inteligência Artificial" (Lug02), pág 192, convenientemente esclarecido e modificado.

   Numa cerimônia de chá muito antiga, da civilização hindu, há três participantes: um ancião, um servente e uma criança. As quatro tarefas que elas executam são:

   - alimentar o fogo
   - servir bolos
   - servir o chá
   - ler poesia

   Esta ordem reflete a importância decrescente das tarefas. A criança começa a desempenhar sua tarefa e no final da cerimônia é o ancião que a encerra. Todas as pessoas executam todas as tarefas. Ninguém pode executar uma tarefa de menor importância dos que as que já tenha executado. Em outras palavras, no instante 1, a criança lê poesia. No instante 2, o servente lê poesia e a criança serve chá. Finalmente no instante 6 o ancião alimenta o fogo.

   (a) Gere as tarefas em cada um dos 6 momentos, conforme as regras acima.

   (b) Escreva uma função recursiva que imprima os momentos acima descritos.

   PS: Antes que você tenha idéias perigosas, imagine o mesmo problema para uma cerimônia ligeiramente maior, com 1830 participantes e uma lista de 879 tarefas.

Exercício da cerimônia hindu

Instante 1 criança-poesia

Instante 2 criança-chá, servente-poesia

3 criança-bolo, servente-cha, ancião-poesia

4 criança-fogo, servente-bolo, ancião-cha

 ©88-09, Pedro Kantek

5 servente-fogo, ancião-bolo

6 ancião-fogo

Para programar isto, crie-se 2 vetores:
pessoa ← criança, servente, ancião
tarefa ← poesia, cha, bolo, fogo

Se olharmos os instantes acima, veremos que no instante 1, a soma dos índices da dupla é 2 No instante 2, a soma é 3 (criança(1)+chá(2) e servente(2)+poesia(1)) No instante 3, a soma é 4 (criança(1)+bolo(3), servente(2)+cha(2), ...)

Desafio

- Escreva um programa que resolva problemas de criptoaritmética.

4.2 Métodos inteligentes de busca

Na aula passada estudamos 6 métodos sistemáticos de efetuar buscas em árvores representando o espaço de estados de um problema qualquer. Lembrando, o que caracteriza um problema deste tipo é:

- um estado inicial
- uma lista de operadores que ao serem aplicados a um estado geram o estado sucessor daquele
- uma condição que se torna verdadeira em um estado meta

Graças ao fenômeno da explosão combinatória, as técnicas da aula passada podem ser muito ineficientes ou até ineficazes para problemas de médio ou grande porte.

Para ajudar a diminuir consumos (tempo e memória) é conveniente introduzir alguma inteligência no método.

A primeira coisa necessária é uma *função de avaliação* que deve examinar um determinado estado qualquer e produzir um número (uma nota) dando conta de quão desejável ou indesejável é a expansão daquele nodo em particular. Quando os nodos disponíveis são ordenados de maneira que o mais desejável seja expandido primeiro, esta estratégia recebe o nome de *Busca pelo melhor*. O nome é algo ambíguo. Na verdade, se fosse possível expandir sempre o melhor, não seria uma estratégia de busca e sim de construir diretamente a solução esperada. O defeito ocorre devido a imperfeições da função de avaliação. Ainda assim o nome se mantém [2]

Uma possibilidade de implementação é tentar estimar (através da função de avaliação) quanto custará para chegar até a meta. Feita esta análise para todos os nodos da borda, será escolhido para expansão aquele que menor valor tiver.

O nome para esta função de avaliação é *heurística* e seu símbolo é h, ou mais apropriadamente h', já que esta heurística é apenas uma aproximação da função que resolve o problema, esta supostamente desconhecida.

A palavra *heurística* deriva do verbo grego *heuriskein* que significa encontrar ou descobrir. Diz-se que Arquimedes, atarantado com um problema posto pelo rei (decidir se um ourives houvera ou não desviado ouro na construção de uma coroa, substituindo-o por prata, sem destruir a peça) teve um estalo quando entrou na banheira e saiu nu gritando pela rua: Eureka (*eu achei!*). Ele acabara de descobrir o Princípio de Arquimedes.

[2] o nome alternativo *Busca pelo que parece ser o melhor* soa algo esquisito.

Em 1957, Polya publicou um livro denominado *How to solve it* onde ele definiu heurística como a arte de descobrir e inventar técnicas para a solução de problemas. Newell, Shaw e Simon usaram em 1963 o termo heurístico como antônimo de algorítmico. Eles escreveram *Quando um processo afirma que resolve determinado problema, mas não oferece nenhuma garantia de fazê-lo, se diz que ele é a heurística para esse problema.* Heurística já foi usada como regras práticas para encontrar soluções sem ter que examinar todo o espaço de estados possíveis. Hoje é usado como adjetivo para qualificar qualquer técnica que melhore o desempenho médio de um resolvedor de problemas.

Outras vantagens do uso de heurísticas:

- Raramente precisamos a solução ótima. há evidencias de que as pessoas são satisfazedoras e não otimizadoras (Simon,81). Exemplo: achar lugar para parar o carro.
- Dificilmente a pior hipótese aparece no mundo real.
- Tentar entender ou melhorar uma heurística resulta em compreensão mais profunda do problema.

A função heurística h' que se usará aqui terá a seguinte definição

h'=custo estimado para ir do nodo atual até a meta

Formalmente falando, h' pode ser qualquer função, desde que o $h'(meta) = 0$. Quanto mais "inteligente"for a função heurística mais eficiente será o metodo de busca que a utilizar.

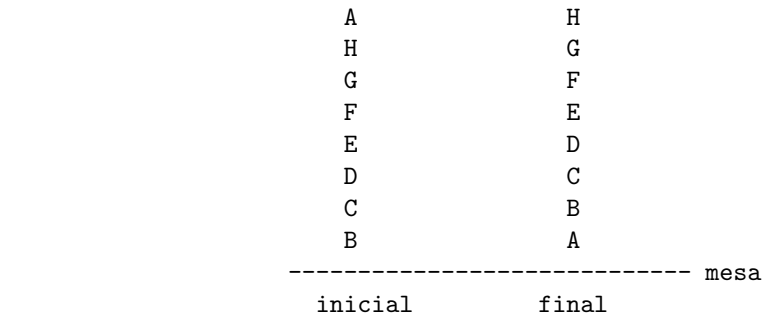
A definição de uma h' adequada sempre depende do problema (e da especificação do problema) que se está a examinar.

Por exemplo, em um problema de dirigir-se de uma cidade a outra, uma boa heurística poderia ser a distância em linha reta entra a cidade atual e a cidade meta.

Existem outras heurísticas mais ou menos gerais. Por exemplo, a seguinte de Lenat normalmente ajuda: Se há uma função interessante de dois argumentos $f(x, y)$, analise o que acontece se x for igual a y.

Significado de f	Resultado da heurística de Lenat
multiplicação	quadrado
união	identidade
contemplar	intrsopecção
matar	suicídio

Seja um exemplo prático Vejamos um exemplo prático desta coisa: Seja o problema de blocos:



Suponhamos que os operadores são:

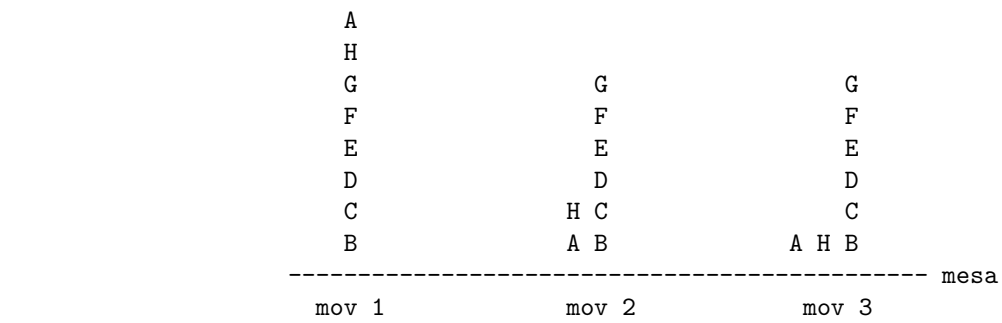
- ponha um bloco livre (não tem nada encima) sobre a mesa

 ©88-09, Pedro Kantek

- ponha um bloco livre sobre outro livre

Usemos a seguinte heurística: Cada bloco que estiver sobre o bloco correto vale 1 ponto. Cada bloco que estiver sobre o bloco errado vale -1. A função heurística é a soma destes valores.

Claramente, o estado meta (final) vale 8. O estado inicial vale 4 (os blocos C, D, E, F, G e H valem 1 ponto e os blocos A e B valem -1). O único movimento possível é colocar o bloco A sobre a mesa, e depois disto a função heurística do segundo estado vale 6. Para o terceiro estado são permitidos 3 movimentos, a saber:



O valor da heurística para os 3 novos estados é 4, talvez esta heurística não seja muito adequada. Tentemos outra heurística:

Quando o bloco tem a estrutura de apoio correta (ou seja, abaixo dele é exatamente como deveria ser) some 1 ponto para cada bloco abaixo. Quando a estrutura de apoio é incorreta, subtraia um ponto para cada bloco abaixo.

Agora o estado final vale 28 (1 para B, 2 para C, 3 para D...7 para o H). O estado inicial vale -28 (tá tudo errado). Mover A para a mesa resulta em -21, já que A deixa de ter os 7 errados abaixo dele. Os valores para os 3 movimentos possíveis são: mov1 = -28; mov2 = -16 e mov3 = -15. Agora o método escolhe o certo que é o mov3.

4.2.1 Exercício

Para cada um dos exercícios a seguir, descreva

- o estado inicial
- a lista de operadores
- a condição de estado meta
- a função de custo
- uma estrutura de dados adequada para guardar o nodo da árvore de busca

9 peças 3 são brancas, 3 vermelhas e 3 pretas, distribuidas aleatoriamente num tabuleiro longitudinal de 10 casas. O problema é ordená-las de modo que as 3 brancas fiquem à esquerda, as 3 negras no meio e as vermelhas à direita. A casa livre pode ficar em qualquer posição. As peças podem ir para a casa livre, com custo igual ao de casas saltadas (se a peça pular para a posição vazia e esta for vizinha, o custo é 1. Se houver 1 no meio, o custo é 2 e assim por diante).

Quadrado mágico Dispor os números de 1 a 9 em um quadrado 3×3 de modo que a soma das linhas, colunas e diagonais seja idêntica.

Jogo de Grundy Dois jogadores tem diante de sí uma pilha de altura arbitrária de moedas. O primeiro jogador divide a pilha original em duas, também arbitrárias. O jogador seguinte escolhe uma das pilhas (deixa a outra intocada) e divide-a. Vão se formando um monte de pilhinhas. O procedimento segue até que cada pilha tenha exatamente 1 ou 2 moedas. Perde o jogador que não puder jogar.

8 bispos Este jogo é pedido por um adventure game para passar para a próxima fase. Trata-se de um tabuleiro usual de xadrez de 5 linhas por 4 colunas. Na linha 1 estão 4 bispos brancos, e na linha 5 estão 4 bispos negros. O objetivo do quebra-cabeça é levar todos os bispos brancos para a linha 5 e os negros para a linha 1, sem que eles se ameacem mutuamente em qualquer instante.

4.2.2 A*

O algoritmo A*, criação de Hart, Nilsson e Raphael apareceu em 1968. Um artigo de 1972, de Hart, corrigiu alguns pequenos equívocos da apresentação original. A idéia é juntar na mesma busca, vantagens da busca gulosa (sempre a opção mais apetitosa) e da busca em largura. A primeira é eficiente, mas pode chegar a um beco sem saída. A segunda sempre chega à resposta, mas pode ser tremendamente ineficiente. Usando A*, a função de avaliação do próximo candidato a expandir na árvore de busca é

$$\boxed{f(n) = k_1 \times p(n) + k_2 \times h'(n)}$$

Sem perda de generalidade, pode-se inicialmente, considerar $k_1 = k_2 = 1$ e com isso a fórmula com que trabalharemos, por enquanto, é:

$$\boxed{f(n) = p(n) + h'(n)}$$

Trocando em miúdos, para cada nodo na árvore, dever-se-á calcular sua profundidade, e também a heurística do nodo. O campeão (ou seja, o nodo que vai ser expandido) é aquele que tiver o menor valor de $f(n)$.

$p(n)$ calcula o valor do custo necessário para ir do nodo inicial até o nodo em questão. Já $h'(n)$ estima o custo necessário para ir do nodo atual até um estado meta. Como o objetivo é minimizar o caminho total, deve-se expandir o nodo que tiver a menor soma de ambas parcelas. Nesse sentido, $f(n)$ pode ser entendido como uma estimativa do menor custo para ir do estado inicial até um estado meta, passando por n.

Se se conseguir impor uma pequena restrição à função $h'(n)$, pode-se afirmar que a busca A* é *completa* e *ótima*. A restrição é de que deve-se escolher uma função h' que nunca sobreestime o esforço necessário para alcançar a meta. Ou seja, a função deve ser otimista: pode errar para menos no esforço que será necessário, mas nunca poderá errar para mais.

Um bom exemplo para esta função, no caso de um problema que visa determinar qual o melhor caminho rodoviário entre duas cidades, poderia ser a distância linear entre cada ponto parcial e o destino. Perceba que esta medida erra para menos, mas nunca pode errar para mais.

4.2.3 Quebra-cabeça 8

Um dos primeiros problemas estudados foi o do QC8. Trata-se de um dispositivo plástico quadrado, com espaço para 9 placas (3×3). Apenas 8 placas são colocadas, o que disponibiliza um espaço que por sua vez permite o movimento das placas vizinhas ao espaço. Dada uma configuração qualquer, e uma determinada configuração final, o quebra cabeça consiste em mover as peças que levam até o estado meta. Uma solução típica consiste de 20 passos, mais ou menos, dependendo da configuração inicial. O fator de ramificação médio é 3 (quando o buraco está no centro são 4 movimentos. Quando está nos 4 cantos, são apenas 2 movimentos. Nas demais 4 casas, são 3 movimentos possíveis).

 ©88-09, Pedro Kantek

Uma busca burra, por exemplo em profundidade com 20 movimentos examinaria $3^{20} = 3.5 \times 10^9$ estados. Se desprezarmos estados repetidos, o espaço de busca diminui, para $9! = 362.880$ possibilidades, já que este é o número de arranjos possíveis em 9 casas.

Para usar A*, há que se encontrar uma heurística que não sobreestime o esforço. Duas possibilidades:

- h_1 = quantidade de peças que estão em lugar incorreto.
- h_2 = soma das distâncias *Manhattan* de todas as peças vis-a-vis seu local correto.

Exercícios

1. Calcule h_1 e h_2 para as seguintes configurações iniciais. Considere a seguinte configuração final

1	2	3
4	5	6
7	8	

3	6	7
1	5	
2	4	8

   Resposta: $h_1 =$ _____, $h_2 =$ _____

2. Calcule h_1 e h_2 para o mesmo estado final.

4		2
6	8	7
3	1	5

   Resposta: $h_1 =$ _____, $h_2 =$ _____

4.2.4 Solução do cubo RUBIK

Por P.Kantek. É muito difícil definir um cubo padrão. Assim, imaginar-se-á um cubo que contém pelo menos a cor azul. A peça a seguir reconhecida como PIVOT é a central de cada face, lembrando que ela determina a cor da face. Além disso, cada face conta com 4 meios (as 4 peças que são vizinhas do pivot) e 4 cantos.

Neste cubo, existirão 6 faces, a saber: frontal e traseira, esquerda e direita e superior e inferior, reconhecidas pelas letras: F, T, E, D, S e I.

Para a rotação, definem-se dois sentidos: o horário e anti-horário. Nos lados esquerdo e direito, imagine que você está olhando de frente a face referida. Para o frontal / traseiro e superior / inferior, você olha ambas de onde você está.

Quando há duas operações de rotação iguais seguidas, tanto faz elas serem hOrário ou Anti-horário. Daí, esta especificação não é escrita.

1. Faça a face azul: primeiro o meio, depois os 4 meios e finalmente os 4 cantos.
2. Acerte os 4 meios azuis com a cor de cada pivot.
3. Acerte as 4 pontas do azul de maneira a que as 3 cores coincidam. Agora, todo o plano azul esta correto.
4. Com a face azul para cima, olhe a face da frente. Para ir de 6h a 3h, faça IO, DA, IA, DO, IA, FO, IO, FA. Para ir de 6h a 9h: IA, EO, IO, EA, IO, FA, IA, FO.

   Ao final, 2 planos estão completos. A seguir, trata-se o terceiro e último plano.

5. Acerto das 4 pontas do último plano. Ponha o azul embaixo, o seu oposto acima. Olhe para a face frontal. Acerte 2 pontas que tem a cor da frente (não importa sua orientação e sim a posição). Para trocar as posições 13h30 com a 10h30: EA, SA, EO, FO, SO, FA, EA, SO, EO, S, S.

   Para trocar o 13h30 com o 13h30 da face direita vista de frente, faça SO, EA, SA, EO, FO, SO, FA, EA, SO, EO, SO.

   Repita estas etapas se necessário. A fase termina quando as 4 pontas estiverem em suas posições (não necessariamente o sentido) corretos.

6. Acertando o sentido das 4 pontas: Aqui existem 7 configurações possíveis. Todas elas sofrem a mesma seqüência de operações. Supondo um cubo só com 3 faces visíveis (a frontal, a direita e a superior) e numerando cada peça como

```
25 26 27                    1,3,27
22 23 24                    10,12
19 20 21                    1,10,12
---------|                  3,19,25
 1  2  3 | 10 11 12         12,21
 4  5  6 | 13 14 15         1,21
 7  8  9 | 16 17 18         3,19,27
```

   e considerando a cor oposta ao azul, localize quais peças (das 27 mostradas ai), são dessa cor. Ache qualquer uma das 7 alternativas de distribuição da cor oposta pelas 27 peças descritas. Achando qualquer uma das 7, aplique: EA,SA,EO,SA,EA,S,S,EO,S,S.

   Note que a peça de quina no cubo acima é a peça 3-10-21. Note também que todas configurações têm o 23, que é da cor oposta. Pode ser necessário fazer até 3 vezes este procedimento.

   Esta fase termina quando as 4 pontas estiverem em posição e sentido corretos.

7. Acertando os 4 meios: Obs: (o meio vertical corresponde ao plano perpendicular a você. O observador está a esquerda, olhando o cubo, logo - e com o perdão pela escatologia - Horário: engula. Anti-horário: vomite.

   Pelo menos 1 dos quatro vai estar certo. Localize-o e ponha-o de frente. Daí execute 1 ou 2 vezes: MVA, SA, MVO, S, S, MVA, SA, MVO.

   Termina com os 4 meios nos seus lugares, ainda que desorientados.

8. Orientação dos meios: Se forem 2 vizinhas trocadas, coloque-as no plano superior de frente para você à direita e: FA, EA, DA, MHO, D, D, MHO, MHO, DA, S, S, DO, MH, MH, D, D, MHA, DO, S, S, EO, FO.

   Se houverem 2 trocadas em situação oposta, coloque-as na fatia vertical do meio paralela a você e: DA, MHO, D, D, MH, MH, DA, S, S, DO, MH, MH, D, D, MHA, DO, S, S.

   Se houverem 4 trocadas, faça primeiro uma e depois a outra.

4.2.5 Resumo

- Faça uma face: primeiro o meio, depois os 4 meios e finalmente os 4 cantos.
- Acerte os 4 meios da face feita com a cor de cada pivot adjacente
- Acerte as 4 pontas da face de maneira a que as 3 cores coincidam. Agora, todo um plano esta correto.

 ©88-09, Pedro Kantek

- Acerte o plano do meio. Com a face feita para cima, olhe uma face de frente e acerte. Agora 2 planos estão completos.
- Acerte as 4 pontas do último plano . A fase termina quando as 4 pontas estiverem em suas posições (não necessariamente o sentido) corretos.
- Acerte o sentido das 4 pontas. Termina quando as 4 pontas estiverem em posição e sentido corretos.
- Acerte os 4 meios.

Desafio

- Implementar o solucionador do quebra-cabeça 8.

4.3 A*, continuação

4.3.1 Mais sobre a heurística

Uma maneira de caracterizar a qualidade da heurística usada é mediante um parâmetro a que se pode chamar de **fator de ramificação efetiva - fre**. Assim, se a quantidade total de nodos expandidos pelo algoritmo A* em um determinado problema é N, e a profundidade da solução é d então fre é o fator de ramificação que deve ter uma árvore de profundidade d que tenha N nodos, ou

$$\boxed{N = 1 + fre + (fre)^2 + ... + (fre)^d}$$

Por exemplo, se A* acha uma solução em profundidade 5 e utiliza 52 nodos, então fre é igual a 1.91. Em geral, o fre de uma determinada heurística se mantém constante através de uma ampla gama de problemas, pelo que seu desempenho pode ser estimado e servir de guia para a utilidade global da heurística. Para uma heurística bem desenhada, o fre deve se aproximar de 1, o que pode abrigar a solução de problemas bastante grandes. Para analisar este fato, geraram-se 100 problemas aleatórios, com longitudes de solução variando entre 2 e 20, que foram resolvidos usando A* com h_1 e h_2 e para efeitos de comparação com a busca por aprofundamento iterativo (vide aula 4). Eis os resultados, extraídos de [Rus96], pág.110, edição espanhola.

	Custo da busca			fator de ramificação efetivo		
d	BAI	$A*(h_1)$	$A*(h_2)$	BAI	$A*(h_1)$	$A*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6.384	39	25	2.80	1.33	1.24
10	47.127	93	39	2.79	1.38	1.22
12	364.404	227	73	2.78	1.42	1.24
14	3.473.941	539	113	2.83	1.44	1.23
16	-	1.301	211	-	1.45	1.25
18	-	3.056	363	-	1.46	1.26
20	-	7.278	676	-	1.47	1.27
22	-	18.094	1.219	-	1.48	1.28
24	-	39.135	1.641	-	1.48	1.26

4.3.2 como inventar funções heurísticas

Como vimos h'_1 e h'_2 são estimativas de quanto falta para solucionar o problema. Entretanto, elas podem ser consideradas funções exatas para o custo de solucionar problemas

mais simples. Se as regras do quebra-cabeça fossem modificadas de maneira que uma peça pudesse se deslocar para qualquer posição, em vez de poder fazê-lo apenas ao quadrado vizinho vazio, então h'_1 daria a quantidade total de passos necessários para resolver o problema. Se a peça pudesse se deslocar apenas um quadrado em qualquer direção, incluindo um espaço já ocupado, então h'_2 daria a quantidade exata de passos até a solução do problema.

Denomina-se problemas *relaxados* a problemas identicos ao original, mas que diminuem as restrições usadas nos operadores. É comum que o custo da solução de um problema relaxado seja uma boa heurística para o problema original.

Quando se escreve a definição do problema original em linguagem formal, é possível construir imediatamente o problema relaxado. Por exemplo, se os operadores do problema QC8 forem:

Uma peça pode ir de A até B se A está junto a B, e se B está vazio

Para gerar 3 problemas relaxados, deve-se eliminar 1 das restrições, ou ambas e fica:

1. Uma peça pode ir de A a B, se A está junto de B
2. Uma peça pode ir de A a B, se B está vazio
3. Uma peça pode ir de A a B

Recentemente, criou-se um programa (ABSOLVER) capaz de gerar novas heurísticas a partir da definição do problema empregando a técnica do problema relaxado, além de outras. Este problema produziu uma nova heurística para o QC8 melhor do que qualquer uma previamente criada e ofereceu a primeira heurística útil para o problema do cubo Rubik.

Suponha que para um dado problema se consiga escrever diversas heurísticas h'_1, h'_2, ... ,h'_m, sem que nenhuma delas seja evidentemente "a melhor". Uma abordagem interessante é fazer:

$$h'(n) = max(h'_1(n), ..., h'_m(n))$$

4.3.3 O papel de k_1 e k_2

Comentários sobre o desempenho de um A* (a-est2.exe) Seja a configuração inicial:

1	2	3
6	5	4
8	7	

com o seguinte resultado

k_1	k_2	heurística	nodos	caminho
1	1	1	5.781	20
1	2	1	1.300	20
1	5	1	527	20
2	1	1	16.573	20
1	1	2	2.248	22
1	2	2	724	22
1	5	2	175	24
2	1	2	5.018	20

Seja a outra inicial:

	7	5
3	1	8
6	4	2

com o seguinte resultado

 ©88-09, Pedro Kantek

k_1	k_2	heurística	nodos	caminho
1	1	1	4.592	20
1	2	1	849	20
1	5	1	161	20
2	1	1	15760	20
1	1	2	168	20
1	2	2	838	30
1	5	2	176	40
2	1	2	2.733	20

Seja a última configuração inicial:

1	4	7
2	5	8
6	3	

com o seguinte resultado

k_1	k_2	heurística	nodos	caminho
1	1	1	93.567	28
1	4	1	29.066	32
2	1	1	152.437	28
1	1	2	13.439	28
1	4	2	523	42
2	1	2	74.150	28

4.3.4 Melhoramento interativo

Certa classe de problemas admite soluções nas quais o caminho do estado inicial até o final é irrelevante. Neste caso, o que interessa é o estado final em si. São exemplos desta classe, a distribuição de áreas em circuitos VLSI ou as 8 rainhas, com a estratégia que contempla alocar já de cara as 8 rainhas.

No primeiro exemplo, dada uma configuração inicial de componentes na placa(possivelmente inadequada) a mesma vai sendo gradativamente melhorada. No caso das 8 rainhas, provavelmente haja inúmeros ataques na configuração inicial. Do mesmo jeito, trata-se de ir deslocando as rainhas até a condição de fim ser alcançada.

Para este tipo de problema, a estratégia é *começar com uma configuração inicial completa e depois ir melhorando-a em busca de qualidade*. Suponha-se que cada configuração possível corresponde a um ponto em uma paisagem qualquer. A altitude deste ponto corresponde a uma valoração daquela configuração. A busca, neste caso, equivale a procurar outros pontos da paisagem que sejam mais "altos"do que aquele onde se está.

Existem diversos métodos de busca por melhoramento iterativo, veremos alguns:

Subida da encosta

Neste caso não há uma árvore de busca. O método não preserva a memória dos locais por onde a busca passou. Apenas interessa o ponto atual. Os problemas que este método apresenta são

Máximos locais Contrariamente a um máximo global, este é um valor rodeado por valores mais baixos, mas que ainda assim não é a resposta procurada. Se nada for feito, o algoritmo pára por aqui.

Planície Áreas onde a função de avaliação retorna o mesmo valor.

cume máximo local especial. Ele é uma área mais alta que seus vizinhos, mas a rampa comparada com os movimentos possíveis impossibilita atravessar a cordilheira em um único movimento.

Nestes casos, a solução é o reinício aleatório do processo, mas sempre guardado o maior valor até então. O fim do processo pode ser dependendo de um certo número fixo de iterações, ou o mesmo pode ser encerrado quando após um certo número fixo de iterações o valor objetivo não tenha sido significativamente afetado. Para espaços "bem comportados"este método quase sempre encontra resultados satisfatórios. Para espaços multi-dimensionais, mais semelhantes a um porco-espinho n-dimensional, o método pode falhar em achar soluções.

Finalmente, é bastante comum usar este método em conjunto com outros, de maneira a introduzir melhoras incrementais no resultado alcançado. Em particular, nos algoritmos genéticos, o uso da subida da montanha é tão comum, que acabou abrindo um novo campo de estudo, denominado *algoritmos meméticos*.

Têmpera (endurecimento) simulada

Neste método, o núcleo é similar ao da subida da encosta. Quando vai se efetuar o movimento, ao invés de escolher o melhor, escolhe-se aleatoriamente um dos possíveis movimentos: Se o escolhido apresentar melhora, ele é executado e torna-se o novo ponto atual. Se ele representar piora, o algoritmo dá um salto com probabilidade exponencial. Trocando em miúdos: quanto menor o salto, maior a probabilidade dele ser executado. Para determinar essa probabilidade, usa-se um segundo parâmetro T. Enquanto T é grande, a probabilidade de grandes saltos é alta. À medida em que T tende a zero, este método passa a se comportar como a subida da encosta. O nome do método vem da analogia com o processo de endurecimento de materiais (vidro, aço...). Nestes quanto mais lentamente se produz o resfriamento, menor a quantidade de energia armazenada na configuração, e mais resistente o material produzido. Na analogia da busca, quanto mais lentamente T diminuir, maior a probabilidade de encontrar o máximo global do problema.

Esta técnica foi usada pela primeira vez nos anos 80 na busca da solução do problema de disposição de componentes em VLSI. Desde então tem sido usado amplamente em problemas de *job shop scheduling* (programação fabril) e outras tarefas de otimização em grande escala, muito estudados em pesquisa operacional.

Busca Tabu

Na busca tabu, dado um determinado ponto candidato à solução, escolhe-se uma vizinhança (que pode ser completa ou parcial – em geral, é parcial quando o custo computacional de calcular o fitness de toda a vizinhança é muito alto) e o melhor dos vizinhos calculados substitui o ponto original. A substituição ocorre mesmo que $f(v) \leq f(s)$ onde $f(v)$ é a aptidão do melhor vizinho e $f(s)$ é a aptidão do ponto original.

Nesse sentido, a busca tabu admite a piora no desempenho da solução, já que este fato é pré-requisito para poder escapar de máximos locais. Entretanto, para evitar um possível retorno a uma solução já gerada, o algoritmo usa o conceito de lista tabu. Esta lista é constituída pelos atributos dos movimentos realizados, de maneira que possa ser facilmente consultada e que impeça a repetição ao infinito de movimentos. Movimentos que tenham estes atributos estão impedidos de ser executados por um tempo T, chamado *tempo tabu*, que obviamente é parâmetro da técnica.

Se a lista tabu ficar muito restritiva, é comum usar um critério de relaxamento, permitindo que um movimento que esteja na lista seja ainda assim feito, desde que atenda ao critério de relaxamento.

Um bom exemplo desta técnica está em http://www.sbmac.org.br/tema/seletas/docs/v5_2/00-Souza.pdf

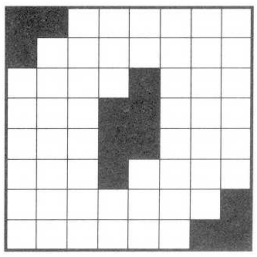
 ©88-09, Pedro Kantek

Problemas de satisfação de restrições

Este tipo de problema desloca um pouco o objetivo da busca. Ao invés de selecionar um nodo final (seja por sua configuração, seja por restrições a que ele obedece) e buscar a seqüência de estados que levam do inicial a este final, nesta classe de problemas, se estabelece que o estado final deve satisfazer a um conjunto de restrições (usualmente representadas em uma estrutura de dados adequada) e a busca se dá por tratativas matemáticas ou heurísticas sobre as restrições. Neste caso não há interesse nos estados intermediários que levaram ao final. Apenas este é importante. Nada impede que se use nesta classe de problemas aos métodos de busca já estudados.

Um bom exemplo desta classe é o problema de assinalamento das n rainhas no tabuleiro de xadrez. Embora este problema (para instâncias pequenas) encontre solução exata e ótima – vide o apêndice no final desta aula – tais métodos não se aplicam, por exemplo quando $n = 1.000.000$.

Outro exemplo desta classe é o da construção de palavras cruzadas. Dada uma matriz como por exemplo

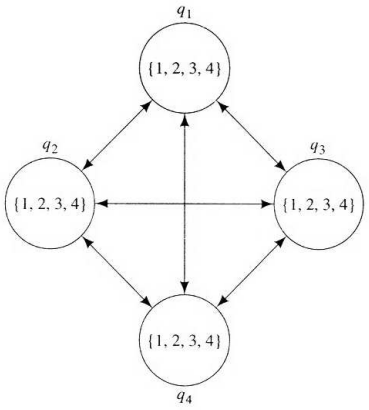


O objetivo é rechear a grade com letras de maneira que as linhas e as colunas formem palavras válidas em um determinado idioma. Este problema, se tratado como busca convencional de estados, tem um espaço imenso. Cada espaço na grade pode recebe 1 de 26 letras. Um estado intermediário poderia ser uma matriz parcialmente preenchida de letras, ainda com espaços em branco. A lista de operadores está em aberto: Pode ser a simples inclusão de uma letra, ou a inclusão de uma palavra completa, ou ainda a substituição de uma letra de uma palavra já colocada. Ginsberg (1990) analisou diversas técnicas de busca neste espaço de estados.

Uma técnica denominada *propagação de restrições* pode ajudar a reduzir o tamanho do espaço de buscas. A técnica é usada junto com um método construtivo, já que a cada passo um novo estado é alcançado. Para estudar a técnica vejamos sua aplicação a uma instância reduzida do problema das rainhas, onde $n = 4$. Para este problema temos 4 variáveis q_1, q_2, q_3 e q_4, que representam as 4 colunas do tabuleiro onde colocaremos as rainhas. Cada uma das 4 variáveis pode receber um de quatro valores $\{1, 2, 3, 4\}$ que correspondem a linha em que a rainha foi colocada. Por exemplo, se a variável q_3 tem o valor 2, isto significa que a rainha foi colocada na coluna 3, linha 2 do tabuleiro. O problema aponta para uma série de restrições entre as variáveis. Por exemplo, se q_1 tem o valor 1, então q_2 não pode ter os valores 1 e 2.

As restrições podem ser descritas em um grafo dirigido, chamado grafo de restrições. Cada nodo representa uma variável, e é rotulado com os valores possíveis para esta variável. Um *arco de restrição dirigido* (i, j) conecta os pares de nodos i e j se os possíveis valores da variável representada no nodo i estão restringidos pelo valor da variável do nodo j.

Na figura a seguir ve-se o grafo de restrições para este problema.



Como todas as variáveis se restringem entre si, todos os nodos tem arcos ligando-os a todos os demais nodos. Diz-se que o arco (i, j) é arco-consistente se para cada um dos valores da variável destino (j) existe ao menos um valor da variável origem (i) que não viola as restrições.

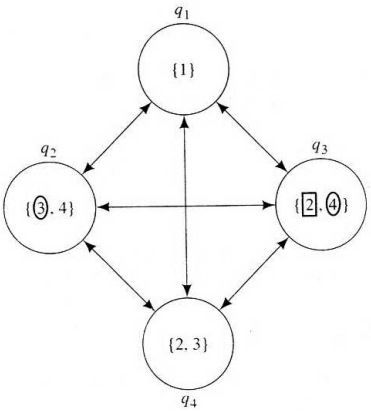
Todos os arcos da figura acima são arco-consistentes, já que para cada par de nodos q_i e q_j (onde $i \neq j$) e para cada valor de q_i existe um valor de q_j que não viola as restrições.

Tendo assinalado um valor a uma ou mais variáveis, pode-se utilizar o conceito de arco-consistência para descartar algum dos valores correspondentes a outras variáveis. Desta maneira o processo de propagação de restrições vai se repetindo em todos os arcos do grafo eliminando os valores no final do arco que não cumprem as restrições. O processo termina quando já não houver valores que possam ser eliminados.

Usando o exemplo, suponhamos um processo de busca originalmente em profundidade. Assim assinale-se o valor 1 à variável q_1 (ou seja, coloque-se uma rainha na coluna 1, linha 1). Acompanhe a propagação de restrições:

- Se considerarmos o arco (q_2, q_1) eliminaremos os assinalamentos $q_2 = 1$ e $q_2 = 2$, já que o valor que fica em q_1 (que acabamos se assinalar) é inconsistente com estes valores.
- O arco (q_3, q_1) elimina os assinalamentos $q_3 = 1$ e $q_3 = 3$ já que estes valores são inconsistentes com $q_1 = 1$.
- O arco (q_4, q_1) elimina os assinalamentos $q_4 = 1$ e $q_4 = 4$.

Obtem-se o grafo seguinte

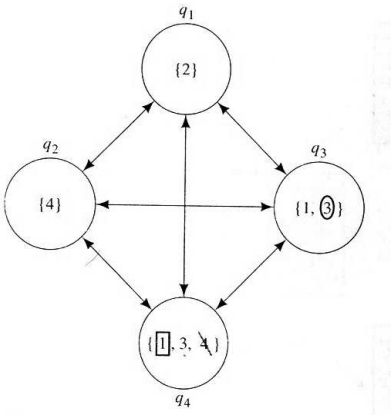


 ©88-09, Pedro Kantek

No passo seguinte, são eliminados todos os valores de q_3, com o que se chega à conclusão de que não há solução possível se $q_1 = 1$. Não há necessidade de continuar buscando mais nodos, com $q_1 = 1$. Volta-se atrás e faz-se $q_1 = 2$. A seguir, o resultado da propagação:

- O arco (q_2, q_1) elimina $q_2 = 1$, $q_2 = 2$ e $q_2 = 3$.
- O arco (q_3, q_1) elimina $q_3 = 2$ e $q_3 = 4$
- O arco (q_4, q_1) elimina $q_4 = 2$

Agora, obtem-se o grafo de restrições para $q_1 = 2$.



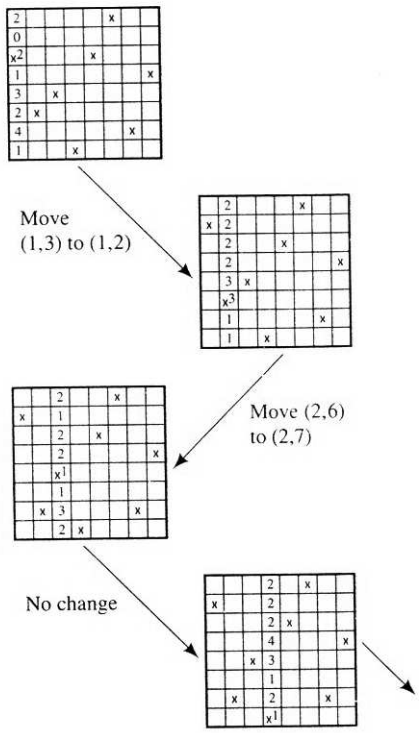
Acabamos de achar uma solução, sem ser necessário pesquisar todos os estados no espaço de estados.

O método de propagação de restrições foi usado em muitos problemas reais (em particular o de rotular linhas na análise de cenas visuais com a identificação $+$, $-$ ou $\rightarrow$. No artigo "Algorithms for constraint-satisfaction problems: a survey"em Artificial Inteligence Magazine 13;1 (32-44), de 1992 há uma revisão dos métodos, extensões e aplicações da propagação de restrições.

Correção heurística

Esta estratégia começa com uma proposta inicial (possivelmente aleatória) que obviamente não satisfaz às restrições. O passo seguinte é ir corrigindo o estado até que as restrições se satisfaçam. Os operadores devem levar a estados diferentes que melhorem a satisfação.

Por exemplo, no problema das rainhas podemos começar colocando uma rainha em cada coluna, em qualquer linha. A seguir, troca-se de posição uma rainha de modo que a configuração final viole menos restrições do que a predecessora. Uma estratégia devida a Gu, (1989), denominada *conflitos mínimos* aplicada a este exemplo, sugere examinar uma coluna por vez, na qual se coloca em cada casa o número total de rainhas que atacam esta casa. A seguir, move-se a rainha desta coluna para a casa que sofre menos ataques (ou que tem o número mínimo de conflitos, daí o nome). Se houver empate, qualquer critério (inclusive o aleatório) pode ser usado para desempatar. Ao fazer isto (provavelmente) o nodo sucessor violará menos restrições. Na figura



pode-se ver uma parte da busca em profundidade para o problema das 8 rainhas.

Para aplicar a correção heurística ao problema das palavras cruzadas pode-se escolher como estado inicial qualquer matriz completamente preenchida com letras aleatórias. A seguir, trocando-se algumas letras da matriz, espera-se chegar próximo de alguma palavra válida nas linhas e colunas. O espaço de busca é muito grande, pelo que o uso de um método construtivo ou uma correção heurística para a manipulação de estados e operadores pode ter efeitos dramáticos na melhoria do processo de busca.

Desafio

- Implementar o problema dos jarros de água.

4.4 JOGOS

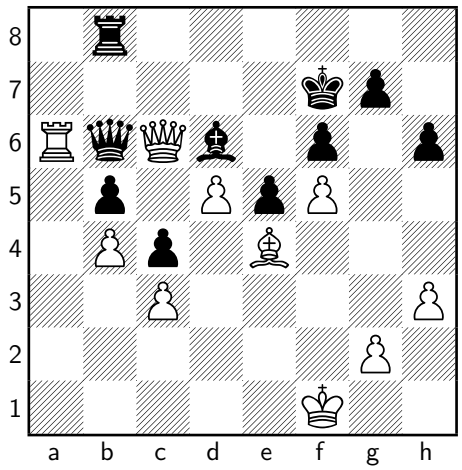
Desde o início da civilização os jogos desempenham papel importante. Jogos de tabuleiro como xadrez, go, damas exercem fascínio por permitirem uma luta abstrata ou seja, é possível guerrear sem ter que organizar exércitos, liderar batalhas e correr riscos. Esse aspecto da abstração é que torna os jogos um bom palco para as estratégias da IA. O espaço é limitado, os operadores finitos (poucos) e conhecidos.

Essa história é antiga. Já em 1950, Shannon e Turing escreveram os primeiros programas de xadrez. Em 53, Turing sugeriu que até o ano 2000, um programa de xadrez ganharia do campeão mundial. Errou por poucos meses: em 4 de maio de 1997[3], Deep Blue jogando com as brancas, derrotou o campeão mundial Gary Kasparov.

[3]No match anterior, em 96, Deep Blue já ganhara uma partida, mas acabou perdendo o match por

 ©88-09, Pedro Kantek

Detalhes do match em http://www.research.ibm.com/deepblue. Veja na figura como estava o tabuleiro quando Kasparov abandonou. Era o lance 45, Deep estava de brancas e Kasparov de negras.



O xadrez já foi descrito como a *drosóphila melanogaster* da IA. Lembrando, a *drosóphila* é a mosquinha caseira usual. Embora seja uma chateação, para os especialistas em genética, essa mosca é um presente dos céus. Ela

- se reproduz em ciclos de 11 dias
- come qualquer coisa
- tem cromossomos enormes, facilmente coráveis e manuseáveis
- não tem nenhum custo

O xadrez foi escolhido por diversas razões. Ao se conseguir que um computador vença uma partida, certamente estará se transpondo um comportamento inteligente a ele. A simplicidade das regras, e o fato de que o jogo implementa o conceito de informação perfeita (o programa consegue ter acesso ao 100% do tabuleiro) permitem que a representação do jogo na memória seja perfeita sem nenhum detalhe perdido.

A presença de um oponente complica o problema na medida em que ela introduz aleatoriedade, já que é impossível saber o que ele vai fazer. Não é a aleatoriedade do clima ou de um dado lançado. Aqui existe a determinação do oponente de impor prejuizo à situação do programa, já que o jogo pode ter apenas um vencedor e o oponente, é óbvio, luta para vencer.

A principal dificuldade reside no tamanho da árvore de possibilidades. Supondo um fator de ramificação médio para o xadrez de 35 (A cada jogada o oponente pode responder de 35 maneiras diferentes) e considerando que um jogo médio tem cerca de 50 lances para cada lado, o tamanho da árvore gerada neste caso seria de 35^{100}. Nestes casos, a incerteza resulta não da falta de informações sobre o ambiente, mas da falta de tempo para analisar todas as alternativas.

Implementações de jogos em computador são obrigadas a penalizar severamente a ineficiência. Essa estratégia gerou muitas boas idéias na programação, quando é necessário obter uma boa resposta, estando a resposta ótima inalcançavel pelo fator tempo.

A apresentação do assunto segue as etapas: (1) Como avaliar quão boa é uma posição. (2) Depois como decidir qual jogada escolher quando houver restrição de tempo e finalmente (3) como integrar as etapas anteriores levando em consideração o inimigo.

4 a 2

Outros jogos

As damas, começaram a ser programadas em 1952, quando o analista Artur Samuel, da IBM, escreveu um programa que aprendia com seu próprio desempenho (jogando contra si milhares de vezes). Jonathan Schaeffer concluiu o jogo chinok [4]. Em 1992, o chinook venceu o Dr Marion Tinsley que fora campeão mundial por 40 anos e que só havia perdido 3 partidas até então. Perdeu a quarta e a quinta. O embate terminou 21.5 a 18.5 pró Marion Tinsley. Um novo match em 1994 foi encerrado por uma doença de Tinsley. O chinook usa um banco de dados de finais com 8 peças ou menos, que possui 443.784.401.247 posições.

4.4.1 Tipos de jogos

Dependendo das características dos jogos, pode-se montar a seguinte tabela

Tipo de jogo	determinístico	de azar
informação perfeita	jogoda velha, damas, reversi, go, xadrez	banco imobiliário, gamão
informação imperfeita	batalha naval	bridge, poker, truco, war

Como observação, os humanos mão querem jogar reversi contra o computador. Este ganha todas. Já o jogo de go, ninguém quer jogar contra o computador porque este joga muito mal. Em go, a quantidade de alternativas a cada jogada ronda o fator 300.

4.4.2 Algoritmo Básico

Suponha-se um jogo de tabuleiro onde os jogadores serão chamados de MAX e MIN. MAX começa o jogo e as jogadas se alternam até que surja o vencedor. O jogo seria representado por

estado inicial que corresponde ao tabuleiro na situação de começo de jogo

operadores a lista de jogadas que o jogador pode fazer a partir de sua situação

condição de fim que determina o fim do jogo

função de utilidade capaz de avaliar uma posição (estado) e atribuir a ele um determinado valor. No xadrez, os resultados podem ser ganhar, perder ou empatar. Isso poderia ser representado por 1, -1 e 0. Caso a árvore não consiga chegar até o fim do jogo, a função de utilidade precisa estimar a qual resultado a situação presente vai conduzir.

Se fosse um problema de busca normal, bastaria escolher uma seqüência de operações que levasse do estado inicial até uma vitória de MAX. Entretanto, esta é a mesma estratégia de MIN, e apenas um deles pode vencer. Assim, o que MAX precisa fazer é bolar uma estratégia que o leve à vitória *independentemente do que fizer MIN*.

4.4.3 Algoritmo minimax

O algoritmo minimax serve para escolher a melhor estratégia para MAX e é composto pelos seguintes passos:

[4] www.cs.ualberta.ca/c̃hinook). Para jogar on-line, um bom endereço é www.darkfish.com/checkers/Checkers.html

 ©88-09, Pedro Kantek

a) Geração de toda a árvore do jogo, até chegar aos estados terminais.

b) Aplicação da função de utilidade a cada estado terminal e obtenção do valor respectivo.

c) Transporte dos valores da utilidade em direção à raiz da árvore, baseado no seguinte critério:

   1 Nodos em que joga MAX: o **maior** dos valores dos filhos deve ser trazido ao pai deles (e conseqüentemente, esta deve ser a jogada feita por MAX)

   2 Nodos em que joga MIN: Não se sabe como MIN jogará, mas deve-se esperar pelo pior e conseqüentemente, deve-se escolher o **menor** valor e levá-lo ao seu pai

d) Finalmente, na raiz, MAX deverá escolher o filho que apresenta maior valor de utilidade e efetuar esta jogada.

Suponha um jogo onde eu (MAX) devo escolher entre 3 jogadas: A_1, A_2 e A_3. Se eu escolher A_1, o meu adversário poderá responder com A_{11}, A_{12} ou A_{13} e assim por diante. Como é um jogo muito simples, só há 2 lances alternados. As nove folhas terminais representam (do ponto de vista de MAX) os seguintes valores: 3, 12 e 8; 2, 4 e 6; 14, 5 e 2.

Quem vai jogar acima do nível terminal é o meu adversário (nodo MIN) e portanto deve subir o menor valor dos filhos (eu imagino que o meu adversário jogará de tal modo que o pior resultado sobre para mim). Assim, sobem 3, 2 e 2. Agora é a vez de MAX jogar e como ele está em um nodo MAX, sobe o maior dos 3 valores, no caso o 3. Este é o lance escolhido por ele.

Este algoritmo apresenta uma imperfeição importante: Usualmente não há tempo para chegar até os nodos terminais da árvore. No jogo de xadrez, se isso fosse possível precisaríamos várias vezes o tempo de existência do universo, para chegar lá. O algoritmo acima serve como bom ponto de partida e eventualmente para a análise matemática de jogos.

Decisões imperfeitas

Como não é possível no caso real mais genérico, chegar até os nodos terminais, deve-se ir até onde for possível (este lugar usualmente é um parâmetro setado em jogos de xadrez contra computador) e trocar a função de utilidade por uma *função de avaliação heurística*.

Interrupção da busca

A maneira mais simples de delimitar a busca é estabelecer uma profundidade determinada. Esta é escolhida de maneira a não estrapolar os limites de tempo oferecidos pelo jogo (ou pela paciência do oponente).

Esta estratégia pode ser catastrófica. Em uma situação (por exemplo) de troca de cavalo por dama, se a interrupção se der no lance de tomada do cavalo, a posição resultante será muito boa para o lado que tomou o cavalo. Não se considerou, por estar fora do limite de eventos, a perda da dama. Se a avaliação tivesse avançado uma posição mais, o resultado seria completamente diferente.

Uma maneira de corrigir esta anomalia é o processo denominado *busca do repouso*. Posições que tem trocado o seu valor material seguem sendo calculadas, mesmo que a profundidade limite tenha sido alcançada.

Hipótese do parceiro perfeito

O algoritmo tem este comportamento. Pode ser interessante alterá-lo. Suponhamos estar numa situação ruim, e temos a jogada A muito ruim, e a B apenas ruim. O algoritmo minimax jogará B. Entretanto, ao lado de A existem muitas opções boas para nós (de modo que se o oponente cometer um erro, nós saimos ganhando), e em B isto não ocorre, e qualquer jogada do adversário não alterará o panorama do jogo. A despeito do que diz o MINIMAX, deveríamos jogar A.

Para poder estimar este tipo de comportamento seria interessante fazer uma análise de quão bom é o nosso parceiro e com isto decidir se dá ou não prá largar uns *facão* encima dele.

Jogo de damas A partir de 1952, Arthur Samuel da IBM, trabalhando em seu tempo disponível, desenvolveu um programa de jogo de damas que apreendeu sua própria função de avaliação jogando consigo mesmo milhares de vezes. O programa de Samuel começou no nível de iniciante, mas, depois de apenas alguns dias jogando sozinho, melhorou seu desempenho além do nível do próprio Samuel (embora Samuel não fosse um bom jogador). Em 1962, o programa derrotou Robert Nealy, campeão na modalidade denominada "jogo de damas às cegas", graças a um erro de Robert. Muitas pessoas acharam que isso significava que os computadores eram. superiores às pessoas no jogo de damas, mas esse não era o caso. Ainda assim, quando se considerar que o equipamento de computação de Samuel (um IBM 704) tinha 10.000 bytes de memória principal, fita magnética como meio de armazenamento de longo prazo e um processador de 0, 00000 1 GHz, a vitória ainda é uma grande realização. Outras pessoas tentaram algo melhor, até Jonathan Schaeffer e seus colegas desenvolverem o Chinook, que funciona em PCs comuns e utiliza a busca alfa-beta. O Chinook usa um banco de dados pré-calculado de todas as posições (444 bilhões) com oito ou menos peças no tabuleiro para tornar seu fim de jogo infalível. O Chinook chegou em segundo lugar no U. S. Open de 1990 e ganhou o direito de desafiar o campeão mundial. Foi então que ele enfrentou um problema, chamado Marion Tinsley. O Dr. Tinsley havia sido campeão mundial por mais de 40 anos, perdendo apenas três jogos em todo esse tempo. Na primeira competição contra Chinook, Tinsley sofreu sua quarta e quinta derrotas, mas ganhou a disputa por 20,5 contra 18,5 pontos. A partida pelo campeonato mundial em agosto de 1994 entre Tinsley e Chinook terminou prematuramente quando Tinsley teve de se retirar por razões de saúde. O Chinook se tornou o campeão mundial oficial. Schaeffer acredita que, com suficiente poder de computação, o banco de dados de finais de jogos poderia ser ampliado até o ponto em que uma busca direta desde a posição inicial sempre chegaria a posições resolvidas, isto é, o jogo de damas seria completamente resolvido. (O Chinook anunciou uma vitória com apenas 5 movimentos.)

4.4.4 Corte α - β

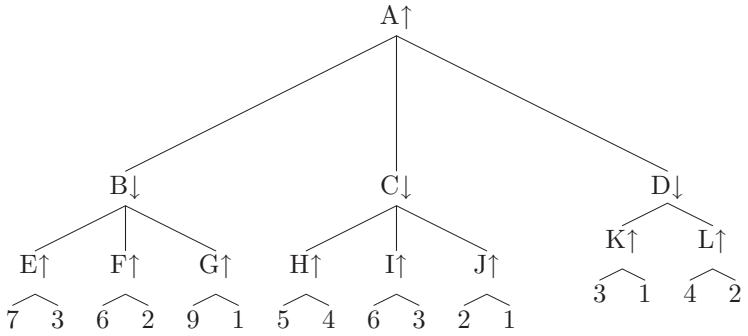
Suponha-se ter implementado uma busca minimax com uma boa função de avaliação para o xadrez inclusive com busca por repouso. Contando com um micro legal, bem programado, o computador poderá calcular 1000 posições / segundo. Qual será o desempenho do programa ? Nos torneios de xadrez, geralmente o tempo médio alocado a um lance é de 2,5 minutos (150 seg). O programa conseguiria olhar 150.000 posições. Com um fator de ramificação 35, o programa conseguirá olhar 3 ou 4 ciclos e seu desempenho será o de um novato. Jogadores humanos médios conseguem prever 6 ou 8 ciclos, e o programa perderia facilmente para estes.

A solução vem do fato de que nem sempre é necessário calcular todos os nodos da árvore para jogar bem. O processo que permite ignorar ramos completos da árvore é denominado *poda da árvore*.

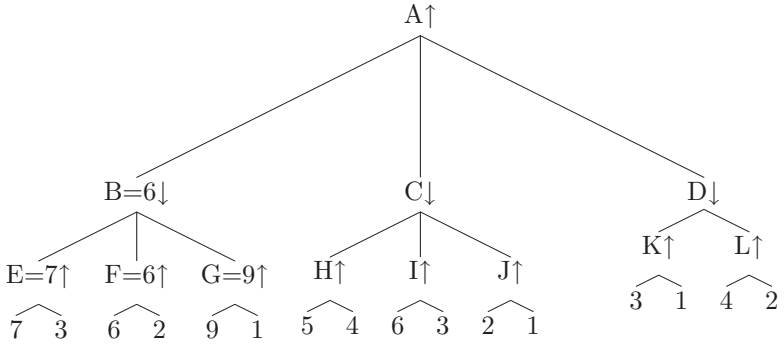
 ©88-09, Pedro Kantek

O corte $\alpha - \beta$ é um método com a capacidade de diminuir o número de nodos que precisam ser examinados durante o procedimento MINIMAX. Não há melhora do resultado alcançado, mas sim no tempo gasto para alcançá-lo. Quando há restrições de tempo (sempre as há), o uso do corte $\alpha - \beta$ permite que se alcance maior profundidade na análise da árvore o que ao fim acaba redundando numa melhora de qualidade do resultado pelo fato da busca ter ido mais fundo na árvore.

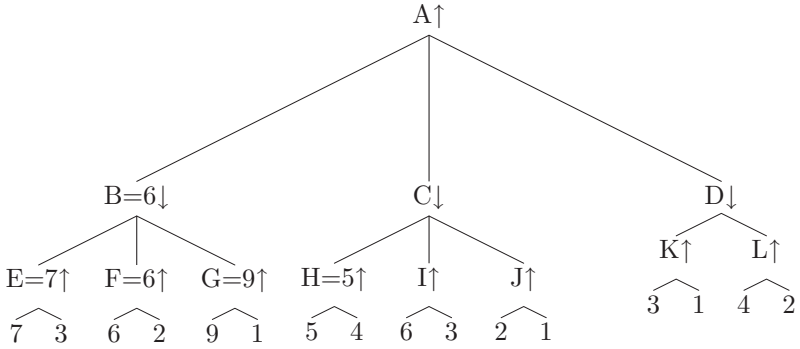
O procedimento começa após ter analisado em profundidade todo a (sub) árvore do primeiro filho à esquerda. Por exemplo, em uma árvore



Após ter percorrido toda a árvore do primeiro filho, fica



A seguir analisa-se o primeiro ramo do segundo filho. Aliás, antes de continuar é bom reforçar este ponto: **o corte $\alpha - \beta$ só começa a funcionar ao se examinar o segundo irmão, já conhecendo o valor do primeiro irmão**. Descendo novamente no primeiro ramo da busca em profundidade do segundo ramo, desce-se até o nodo "H" e ve-se que este vale 5.



Agora, analisemos o nodo "C". Se algum dos irmãos de "H" retornar um valor maior do que 5, este valor será desprezado (pois, "C" é um nodo de mínimo, e portanto, só nos interessam valores menores do que 5). Entretanto, se qualquer valor retornado for menor do que 5, também não interessa, uma vez que, na análise do nodo "A", queremos

valores 6 ou maior do que 6 (por causa do nodo B, e lembrando que "A" é nodo de MAX). Ou seja, se correr o bicho pega e se ficar o bicho mata. Não há necessidade de evoluir os nodos "I" e "J", **nem qualquer dos seus sucessores**, se houverem.

Na verdade, neste ponto da análise, a árvore cuja raiz é "C" pode ser integralmente desprezada.

Por idêntica razão, a sub-árvore "D" também pode ser desprezada e o nodo "A" vale 6. Se não considerarmos o corte $\alpha - \beta$, o número de testes a efetuar é 12 (são 12 nodos, de "A" até "L"). Usando o corte, são apenas 6 testes (se bem que há um ligeiro *overhead* aqui, para se testar os valores de α e de β. Entretanto, deve-se lembrar que a árvore de exemplo tem apenas profundidade = 3. Quanto maior a profundidade da árvore, maior é a economia de tempo obtida pelo corte.

Outro ponto importante é que a ordem de valor dos nodos folha tem uma implicação significativa no desempenho do corte. O ideal é que para filhos de um nodo MAX, estes sejam gerados em ordem decrescente de valoração, e conseqüentemente, para filhos de nodos MIN, os filhos de menor valor sejam gerados antes dos filhos de maior valor. O melhor sempre, é que o gerador gere antes os nodos que tem mais possibilidade de serem os melhores, ou seja os escolhidos. Se isto sempre for possível, a complexidade passa de $O(r^p)$ (r=ramificação, p=profundidade) para $O(r^{p \div 2})$, ou seja, no caso do xadrez acima, a ramificação passa de 35 para $\sqrt{35} = 6$. Ao gerar 150.000 movimentos, o programa pode analisar 8 jogadas em vez das 4 sem o α - β.

No pior caso (distribuição pessimista de valores de filhos), o corte $\alpha - \beta$ pode não economizar nada. No melhor caso, cerca do dobro de profundidade pode ser alcançado.

O algoritmo deste corte foi proposto no final dos anos 50, visando melhorar o desempenho de minimax. Nele, dois valores: α e β são criados durante a busca. O valor α é associado aos nodos de MAX não pode decrescer, e o valor β associado aos nodos de MIN não pode aumentar. Suponhamos que o valor α para um nó MAX seja 10. Então, MAX não precisa considerar qualquer valor menor ou igual a 10 que esteja associado com qualquer nodo MIN que esteja abaixo deste MAX. Identicamente, se MIN tem um valor β de 8, ele não precisa considerar qualquer nodo MAX abaixo que tenha um valor 8 ou mais.

Esta é a maneira conceitual de olhar e entender o algoritmo. Agora vamos acompanhar o próprio.

4.4.5 Algoritmo $\alpha - \beta$

Este algoritmo retorna o valor de avaliação do estado em questão. Alfa é o maior valor ao longo do caminho deste estado (até a raiz). Beta é o menor

```
1: inteiro função avaliamax (estado, alfa, beta)
2: se o estado é uma folha então
3:    retorne o valor deste estado
4: senão
5:    para cada filho s do estado faça
6:       alfa ← maior entre alfa e avaliamin(s, alfa, beta)
7:       se alfa ≥ beta então
8:          ==== CORTE BETA ====
9:          retorne beta
10:         saia
11:      fimse
12:   fimpara
13:   retorne alfa
14: fimse
15: fim-função
```

 ©88-09, Pedro Kantek

```
1: inteiro função avaliamin (estado, alfa, beta)
2: se estado é folha então
3:    devolva o valor da folha
4: senão
5:    para cada filho s de estado faça
6:       beta ← menor entre beta e avaliamax(s, alfa, beta)
7:       se beta ≤ alfa então
8:          ==== CORTE ALFA ====
9:          retorne alfa
10:         saia
11:      fimse
12:   fimpara
13:   retorne beta
14: fimse
15: fim-função
```

Desafio

- Implemente o jogo do fubá.

4.5 O jogo do xadrez

4.5.1 O tabuleiro

O xadrez é um jogo de 2 adversários, jogado sobre um tabuleiro formado por 64 casas, alternadas entre as cores branco e preto. A maneira correta de dispôr o tabuleiro é deixando as casas brancas à direita dos dois jogadores

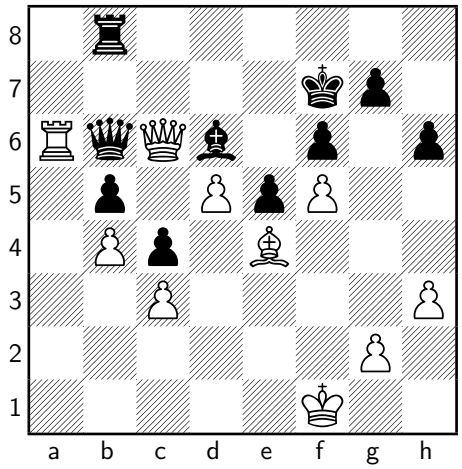
Existem dois jogos de peças, em na cor branca e outro na cor negra. Na representação do tabuleiro em textos (na vertical), o jogador das brancas fica abaixo e o das negras fica acima.

Para representar (e posteriormente estudar) partidas, há a necessidade de identificar cada uma das 64 casas do tabuleiro. Existem diversas maneiras de fazer isto, mas aqui usar-se-á a notação algébrica reduzida (também conhecida como SAN = *short algebric notation*).

Nela, o tabuleiro é dividido em linhas (de esquerda à direita de ambos os jogadores) e colunas que são as filas que vão de um jogador até o outro. As colunas são conhecidas pelas letras latinas minúsculas, começando com a letra **a** à esquerda do jogador de brancas.

As linhas são numeradas de 1 a 8, sendo a linha 1 a mais próxima do jogador das brancas e a linha 8 a mais próxima do jogador das negras.

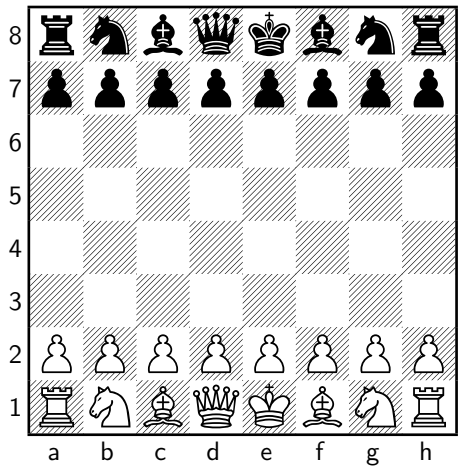
Eis como se representa o tabuleiro:



4.5.2 As peças

cada jogador possui um exército composto de 8 peões, 2 torres, 2 cavalos, 2 bispos, 1 rei e uma dama, totalizando 16 peças. Do ponto de vista de cada jogador, as peças são assim colocadas: na linha mais próxima ao jogador: a torre, um cavalo, um bispo, o casal real (sendo a dama sempre na casa de sua cor), depois um bispo, um cavalo e uma torre.

Na linha seguinte dispõe-se os 8 peões, resultando no seguinte diagrama



4.5.3 O jogo

As brancas sempre começam o jogo, e depois negras e brancas vão se alternando. Com exceção de um lance chamado roque, a cada lance apenas uma pedra pode ser movimentada. Uma peça pode ocupar uma casa originalmente vazia ou então uma casa que esteja ocupada por um adversário, caso em que ele *toma* a peça do adversário após o movimento. Somente a peça que vai ser jogada pode ser tocada. Se se precisar tocar uma peça que não vai ser jogada, usa-se a fórmula francesa *j'addoube*

O objetivo único do jogo é tomar o rei adversário, quando então o mesmo (o adversário) perde.

Os movimentos das peças são:

 ©88-09, Pedro Kantek

rei pode se mover 1 casa em qualquer uma das oito direções possíveis, desde que não ocupe nenhuma casa ameaçada pelo adversário

dama pode se mover quantas casas quiser em qualquer uma das oito direções possíveis, desde que não existam peças pelo caminho. Se forem adversárias, a dama poderá tomá-las. Se forem da mesma cor da dama, elas bloqueiam o movimento. Por exemplo, no início do jogo, as duas damas estão completamente bloquedas.

torre pode se mover quantas casas quiser na fila ou na coluna em que a torre estiver.

bispo pode se mover quantas casas quiser nas diagonais que se cruzam na casa onde o bispo está. Note-se que dos dois bispos, um deles anda só em casas brancas e o outro só em casas negras.

cavalo o cavalo é a única peça que pode saltar sobre peças amigas ou inimigas. Seu movimento simula um "L", já que ele anda duas casas na linha ou coluna onde estiver e mais uma casa em diagonal. O cavalo potencialmente pode fazer 8 movimentos distintos.

Seja um cavalo no tabuleiro de xadrez. Supondo uma matriz de 5×5, e imaginando que o cavalo ocupe a posição central (a 3,3), os saltos que ele pode dar são os seguintes (numerados de 1 a 8).

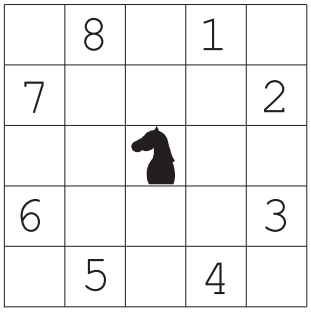


Figura 4.2: Oito movimentos possíveis ao cavalo que esteja na posição 3,3

peão pode andar uma unica casa para a frente. Enquanto estiver na sua posição inicial (e só nesta) poderá andar 2 casas. O peão toma seu adversário de lado. O peão que conseguir chegar ao final do tabuleiro será promovido e poderá se transformar em qualquer peça, a critério do jogador que o promoveu. A nova peça ocupará a peça onde o peão foi promovido. A promoção é notada com um "=", após o movimento do peão, seguido pelo símbolo da peça na qual o peão promovido se transforma.

Uma observação: É comum, após a notação de um determinado lance, aparecem sinais de exclamação e interrogação, com os seguintes significados:

sinal	significado
!	lance bom
!!	lance ótimo
?	lance ruim
??	lance péssimo
!?	lance interessante
?!	lance duvidoso

Roque

O roque é o único movimento do xadrez que permite alterar a posição de duas peças, que são sempre o rei e uma torre. Existe o roque pequeno (o rei e a torre do rei são movimentados) e o roque grande (o rei e a torre da dama são movimentados). O resultado do roque pequeno é que o rei branco vai para a casa **g1** e a torre do rei vai para a casa **f1**. Nas negras, o roque pequeno leva o rei para **g8** e a torre do rei para **f8**.

Já o roque grande, leva (nas brancas) o rei para **c1** e a torre da dama para **d1**, e nas negras o rei para a casa **c8** e a torre da dama para a casa **d8**. O roque pequeno é representado por "0-0"e o grande por "0-0-0".

Os roques só podem ser feitos nenhuma destas condições estiverem presentes

- O rei ou a torre tenham sido movidos previamente
- exista alguma pedra entre o rei e a torre
- exista alguma casa por onde o rei a torre se movimentarão que esteja ameaçada pelo adversário
- o rei esteja em cheque

Cheque

Quando o rei adversário é ameaçado, é obrigatório avisar "cheque"(representado na notação por um sinal de mais "+"após o lance). Quando um rei é ameaçado ele obrigatoriamente deve ser protegido (sair do cheque). Isto pode ser feito de 3 maneiras

- movendo o rei ameaçado
- interpondo um peça qualquer entre a peça que deu o cheque e o rei (é claro, desde que a peça ameaçadora não seja um cavalo)
- tomando a peça que deu o cheque

Se o rei não puder sair do cheque, diz-se então ter sido dado um cheque mate (representado por duas cruzes) e então o jogo acaba com a vitória de quem deu o cheque mate.

Abandono

O jogo não precisa obrigatoriamente seguir até um cheque mate ou empate. A qualquer momento, qualquer um dos dois oponentes podem abandonar o jogo, desde que não visualizem como vencê-lo ou sequer empatá-lo. No tabuleiro este fato é representado quando o abandonante derruba seu próprio rei na sua hora de jogar.

Empate

O empate ocorre nas seguintes circunstâncias:

- pode ser oferecido a qualquer momento por qualquer um dos contendores. Cabe ao oponente aceitar ou não. Além deste caso que é opcional, todos os demais são obrigatórios.
- quando nenhum dos jogadores possui material para vencer o jogo (por exemplo, rei contra rei, rei e bispo contra rei, rei e cavalo contra rei).
- quando o rei é afogado: o único movimento que resta a um jogador deixaria o seu próprio rei em cheque.

 ©88-09, Pedro Kantek

- cheque perpétuo: Um jogador (usualmente o que está superiorizado) não consegue escapar de cheques sucessivos do adversário.

- tripla repetição: Se uma dupla de jogadas for repetida 3 vezes.

- cinqüenta lances: durante o desenvolvimento da partida, 50 lances forem feitos sem que ocorram movimentos ou capturas de peões.

Contagem em matches

Um campeonato de xadrez, usualmente denominado *match* é um conjunto de partidas. Cada vitória rende 1 ponto ao vencedor e zero ao perdedor. Empates determinam meio ponto a cada um dos contendores. Usualmente o match é jogado até um determinado número de pontos, quando se declara o vencedor ou o empate.

4.5.4 Notação

A anotação de um lance se faz escrevendo a letra inicial da peça movida, seguido da casa para onde a peça foi movida. A exceção é o peão, que quando movimentado não deve ser identificado por P (de peão).

Exemplos:

Cf3 - Cavalo efe três. O cavalo foi para a casa "f3".

Dg4 - Dama gê quatro. A dama foi para a casa "g4".

e4 - o peão foi para a casa "e4"

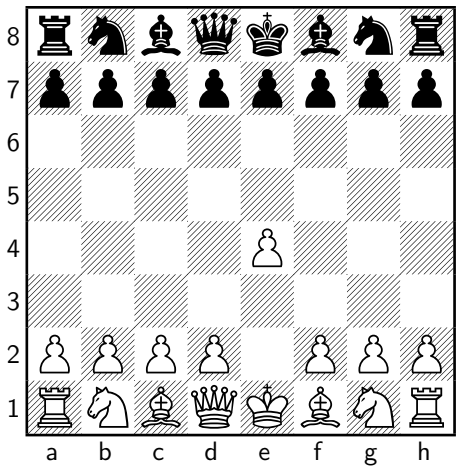
A captura é representada colocando um "X"entre a peça jogada e a casa onde foi feita a captura.

Exemplo:

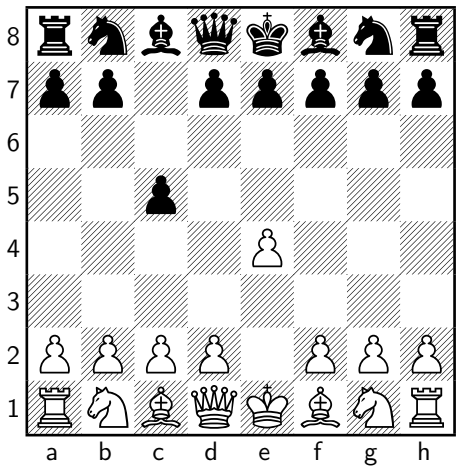
Cxf3 - Cavalo por f3. O cavalo capturou a peça que estava em f3.

Quando houver ambigüidade num certo movimento (por exemplo, os dois cavalos podendo atingir a mesma casa, ao se identificar a peça que se move, escreve-se a sua coluna (ou linha) antes de indicar para qual casa ele se move.

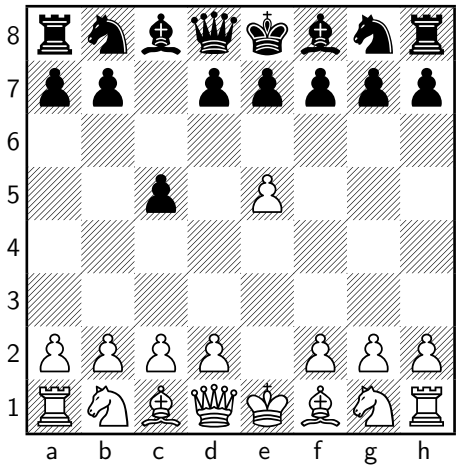
Quando um peão tomar outro *en passant*, anota-se o movimento com a observação "ep"logo após. Veja-se o exemplo: **1 e4**



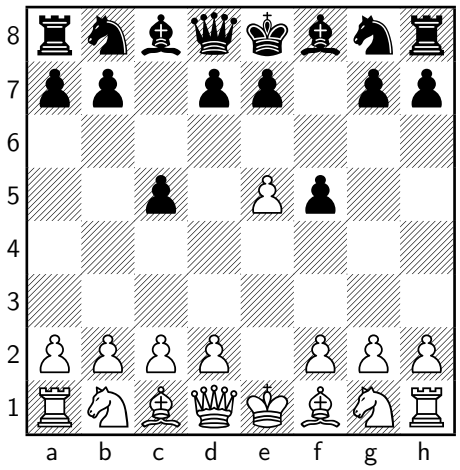
1. . . c5



2 e5

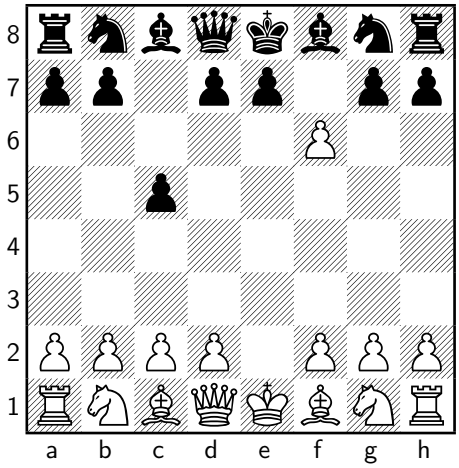


2. . . f5



3 e×f6

 ©88-09, Pedro Kantek



Uma observação final

A notação algébrica em português usa as seguintes letras: P=peão, T=torre, C=cavalo, B=bispo, D=dama e R=rei. A notação algébrica em inglês, que tem a sua importância por ser a interface usual em livros estrangeiros, softwares de xadrez, etc etc é um pouco diferente e usa: P=peão (pawn), R=torre (rook), B=bispo (bishop), N=cavalo (knight), Q=dama (queen), K=rei (king)

Etapas

O jogo de xadrez costuma ser dividido em 3 fases, cada uma com suas peculiaridades.

abertura compreende algo como os primeiros 10 lances do jogo. Visa sobretudo desenvolver as peças e estabelecer uma posição no tabuleiro. Lembremos que as torres, os bispos, a dama e o rei estão no início do jogo completamente bloqueados pelas peças da sua cor.

meio o meio-jogo é a conseqüência da abertura e visa (a) conseguir vantagem material; (b) criar debilidades no campo inimigo e (c) estabelecer combinações (ataques orquestrados de diversas peças). Em geral esta fase assiste a trocas sucessivas de peças, levando ao

final com o tabuleiro já quase vazio, o final assiste ao embate final entre os 2 reis. Ganha importância grande o peão (passado) ou a disposição de peões que permita passar algum, pela sua substituição por peça mais poderosa. Divide-se em finais "ganháveis" (rei e dama contra rei, rei e torre contra rei, entre outros) e finais "não ganháveis" (rei e cavalo contra rei, rei e bispo contra rei, entre outros)

As estratégias de consecução variam em cada uma das 3 etapas: na abertura, grande parte dos movimentos é dicionarizado constituindo-se as milhares de aberturas conhecidas, sendo que as mais famosas ganham até nome: Ruy Lopez, dos 4 cavalos, ...

Já no meio-jogo, a estratégia em estado bruto aparece. Há espaço no tabuleiro, as peças estão mais desenvolvidas. Armadilhas, blefes, tudo vale para ganhar uma posição melhor, alguma qualidade, ou preparar o bote. Muitas partidas terminam aqui, quando não há defesa possível a alguma posição. Entretanto, se os jogadores tiverem nivel equivalente esta fase é reconhecida pelo troca-troca de peças.

No final, um conjunto de regras (orientações) estabelece o que o jogador com a posição melhor deve fazer para obter mate. Outro conjunto de regras auxilia quem está pior.

De novo, há necessidade de planos e de talento. As regras fornecem apenas orientações gerais.

Com a licença pela exagerada simplificação, uma boa estratégia de jogo poderia ser:

fase	estratégia
abertura	objetivo (ou sub-objetivos) de posições e de desbloqueio de peças estabelecidos e busca de lances dicionarizados
meio-jogo	estratégias para obter posição, qualidade (ou até o jogo). Ênfase em uma boa função de avaliação e um bom algoritmo de busca em profundidade para lances selecionados
final	conjunto de regras a seguir, conforme a configuração do final

Esta estratégia, "mista" que envolve diversas estratégias complementares, chamadas em função do jogo em questão, acabou se mostrando a estratégia vencedora desde a década de 60, até os nossos dias.

A diferença entre o computador e um campeão

No xadrez, há um número grande de movimentos estúpidos ou simplesmente sem sentido. A boa heurística é a que mostra um número tratável de possibilidades interessantes. Um bom contraponto a esta última afirmativa mostra como é quando alguém é dotado de habilidade heurística: "o grande Ricardo Reti (um jogador de xadrez) dramatizou a natureza de sua maestria quando alguém lhe perguntou quantos lances previa: "Um", respondeu, "o certo.

4.6 Jogadores automáticos de xadrez

Esta é uma antiga tradição em IA: jogar xadrez. Desde Shannon (Programming a computer for playing chess, 1950) a Turing (Can a machine think, 1951) espera-se do jogo um excelente ambiente para desenvolvimento de teorias, técnicas e métricas da IA. O furor causado pelo match Deep Blue x Kasparov em 1997 ilustra também a importância do xadrez no panorama da IA.

Na tabela a seguir, extraída de Russel 96 (1^a edição em língua espanhola, pág. 146) a pontuação obtida por campeões humanos e automáticos ao longo dos últimos anos

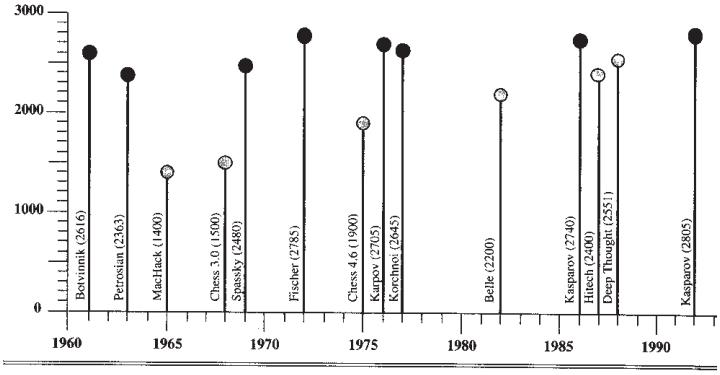


Figura 5.12 Calificación de campeones ajedrecistas, humanos y máquinas.

O primeiro programa capaz de jogar com humanos foi o MacHack 6 de 1967. Conseguiu uma avaliação FIDE de 1400, melhor que a de um iniciante, mas ainda bastante

 ©88-09, Pedro Kantek

longe da necessária para jogar "a sério". Para avaliar o número FIDE imagine uma escala onde o iniciante consegue 1000 pontos, enquanto o campeão mundial tem por volta de 2800 pontos.

Em 1970, começa o ACM North American Computer-Chess Championship, campeonato americano de programas. Em 1974 começa o World Computer Chess Championship. A propósito o primeiro foi disputado em Estocolmo e foi vencido pelo Kaissa, um programa russo. Ele é disputado em média, a cada 4 anos.

Saindo do estado básico (minimax, com alfa-beta, início dicionarizado e final baseado em regras) caso em que os jogadores eram algo melhor que iniciantes, o primeiro grande desenvolvimento se deveu não ao software, mas ao hardware. **Belle** o primeiro jogador baseado em hardware especialmente desenhado foi criado em 1982. Nele tanto o gerador de jogadas quanto o avaliador estavam implementados em circuitos especializados. Belle conseguiu 2.250 pontos, o equivalente ao nivel de mestre.

Em 1980, estabeleceu-se o prêmio Fredkin: 5000 dólares ao primeiro programa que conseguisse pontuação de mestre, (ganho por Belle em 1983), 10000 dólares ao primeiro programa a alcançar 2500 pontos, pontuação próxima à Grande mestre, (ganho por Deep Thought em 1989) e 100000 dólares ao primeiro programa vencedor do campeão mundial humano (Deep Blue em 1997)

HITECH criado pelo ex-campeão mundial de xadrez postal Hans Berliner e por Carl Ebeling também tinha hardware especializado e um avaliador extraordinariamente complexo. Conseguia examinar 10 milhões de posições por jogada. Foi o campeão mundial de programas de 1985 e em 87 foi o primeiro programa a ganhar de um grande mestre humano (Arnold Denker) quando este era um dos 800 melhores jogadores do mundo.

Deep Thought foi originalmente desenvolvido na Universidade de Carnegie Mellon. Rapidamente a IBM contratou parte da equipe e esta lançou o Deep Thought 2: este utiliza um avaliador simples, mas examina cerca de 5 bilhões de jogadas, o que lhe permite ir às profundidades 10 ou 11 do jogo. Em algumas posições, ele avança mais (certa ocasião, achou um mate em 37 jogadas). Nessa época sua pontuação foi FIDE de 2600, o que o colocava entre os 100 melhores enxadristas do mundo.

Deep Blue

O **Deep Blue** foi desenvolvido por Murray Campbell, Feng-Hsiung Hsu e Joseph Hoane na IBM, tomando como base o projeto Deep Thought, desenvolvido anteriormente por Campbell e Hsu na Carnegie Mellon. A máquina vencedora era um computador paralelo com 30 processadores IBM RS/6000 executando a "busca de software"e 480 processadores VLSI especializados para xadrez, que executavam a geração de movimentos (incluindo a ordenação de movimentos), a "busca de hardware"para os últimos níveis da árvore, e também a avaliação de nós folhas. O Deep Blue buscava 126 milhões de nós por segundo em média, com uma velocidade máxima de 330 milhões de nós por segundo. Ele gerava até 30 bilhões de posições por movimento, alcançando rotineiramente uma profundidade igual a 14. O coração da máquina é uma busca alfa-beta de aprofundamento iterativo padrão com uma tabela de transposição, mas a chave de seu sucesso parece ter sido sua habilidade de gerar extensões além do limite de profundidade para linhas suficientemente interessantes de movimentos forçados. Em alguns casos, a busca alcançou uma profundidade de 40 jogadas. A função de avaliação tinha mais de 8.000 características, muitas delas descrevendo padrões de peças altamente específicos. Foi usado um "livro de aberturas"com aproximadamente 4.000 posições, bem como um banco de dados de 700.000 jogos de grandes mestres a partir do qual podiam ser extraídas recomendações consensuais. O sistema também utilizava um grande banco de dados de finais de jogos com posições resolvidas, contendo todas as posições com cinco peças e muitas com seis

peças. Esse banco de dados tem o efeito de estender de forma significativa a profundidade efetiva da busca, permitindo ao Deep Blue jogar com perfeição em alguns casos, até mesmo quando está a muitos movimentos de distância do xeque-mate.

O sucesso do Deep Blue reforçou a ampla convicção de que o progresso dos jogos de computadores vinha principalmente de um hardware cada vez mais poderoso - uma visão encorajada pela IBM. Por outro lado, os criadores do Deep Blue afirmam que as extensões à busca e a função de avaliação também eram críticos. Além disso, sabemos que vários aperfeiçoamentos algorítmicos recentes permitem que programas executados em computadores pessoais padrão vençam todos os campeonatos mundiais de xadrez por computador desde 1992, muitas vezes derrotando oponentes maciçamente paralelos que poderiam buscar um número de nós 1.000 vezes maior. Diversas heurísticas de poda são usadas para reduzir o fator de ramificação efetivo a menos de 3 (comparado ao fator de ramificação real de aproximadamente 35). A mais importante delas é a heurística de movimento nulo, que gera um bom limite inferior sobre o valor de uma posição, usando uma busca rasa na qual o oponente chega ao dobro de movimentos no início. Esse limite inferior freqüentemente permite a poda alfa-beta sem o custo de uma busca em profundidade total. Também é importante a poda de futilidades, que ajuda a decidir com antecedência que movimentos causarão um corte beta nos nós sucessores.

A equipe do Deep Blue recusou a oportunidade de uma revanche com Kasparov. Em vez disso, a mais importante competição recente apresentou em 2002 o programa FRITZ contra o campeão mundial Vladimir Kramnik. A competição de oito jogos terminou empatada. As condições da competição eram muito mais favoráveis ao ser humano, e o hardware era um PC comum, não um supercomputador. Ainda assim, Kramnik comentou: "Agora ficou claro que o principal programa e o campeão mundial são aproximadamente iguais."

Segundo Russel e Norvig, após o match entre DB e kasparov, o valor das ações da IBM aumentou em 18 bilhões de dólares. (Só uma comparação prosaica: se uma ferrari vermelha maravilhosa custa US$300.000, com este valor é possível comprar 60.000 ferraris vermelhas maravilhosas).

4.6.1 A Sexta partida Deep Blue X Kasparov

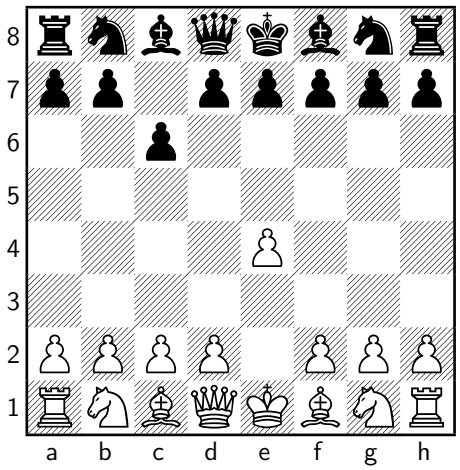
A sexta partida do match foi jogada em Nova York no dia 11 de maio de 1997. Começou as 15h00 e foi a última partida do match que até então estava empatado em 2.5 a 2.5. Kasparov havia ganho o primeiro jogo, perdeu o segundo (abandonando-o) e empatando os jogos 3, 4 e 5, com posições ligeiramente vantajosas para Kasparov nos 3 jogos.

Este jogo atraiu a atenção do mundo, além de ser o último e de ser ganho pelo DB, houve o fato de Kasparov ter desistido no lance 19, após pouco mais de 1 hora de jogo.

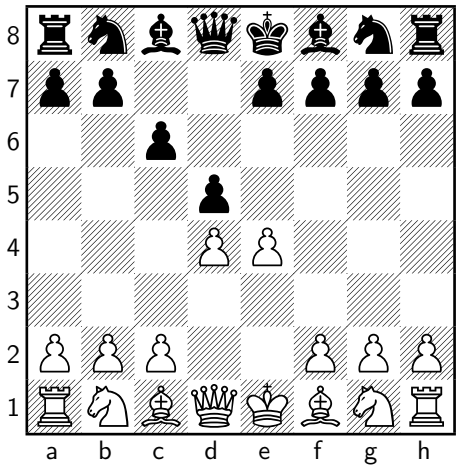
Eis os lances e as principais posições. Deep Blue joga de brancas e Kasparov de negras.

 ©88-09, Pedro Kantek

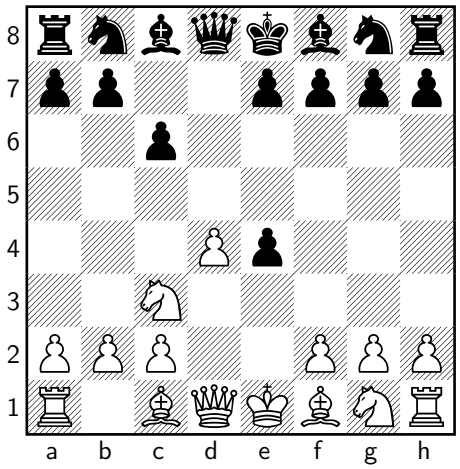
1 e4 c6



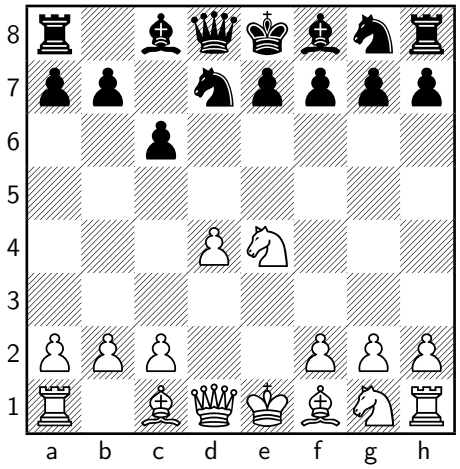
2 d4 d5



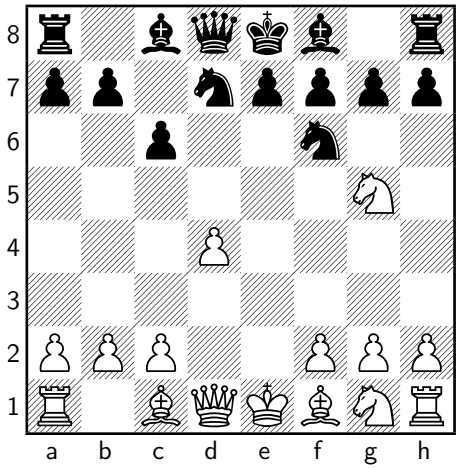
3 ♘c3 d×e4



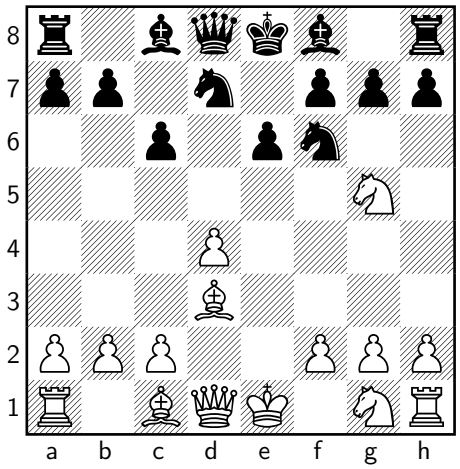
4 ♘×e4 ♘d7



5 ♘g5 ♘gf6

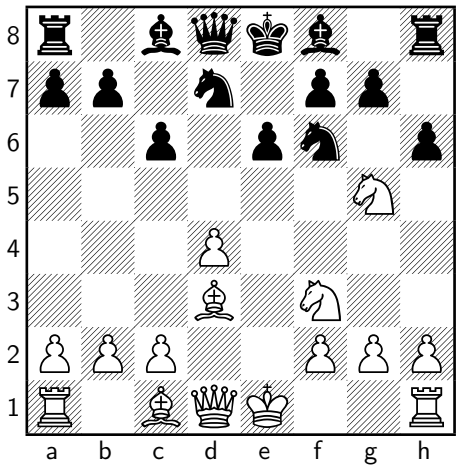


6 ♗d3 e6

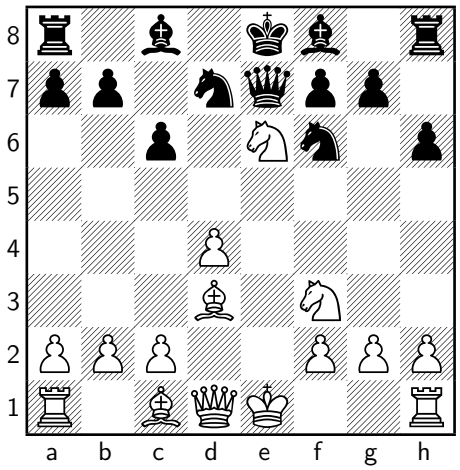


 ©88-09, Pedro Kantek

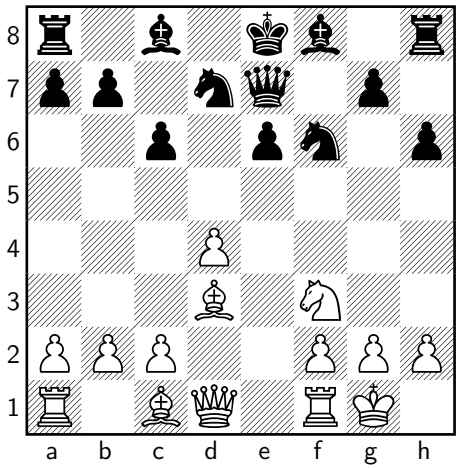
7 ♘1f3 h6?



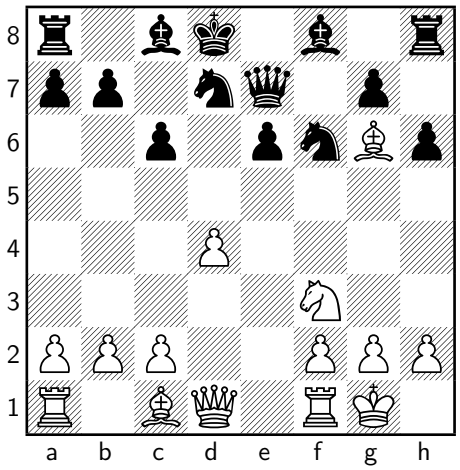
8 ♘×e6! ♕e7



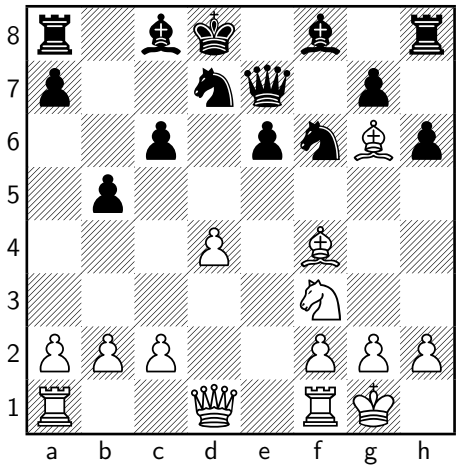
9 O-O f×e6



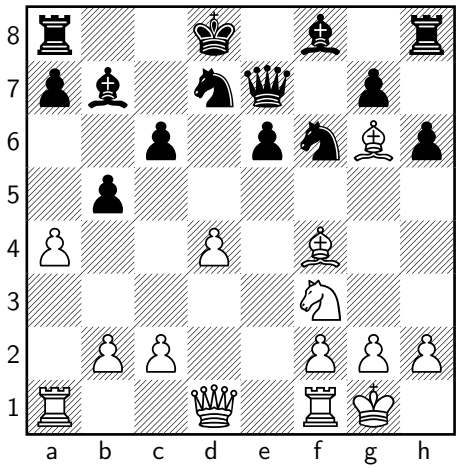
10 ♗g6+ ♔d8



11 ♗f4 b5

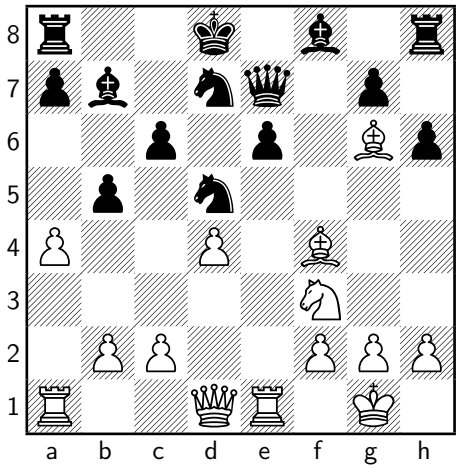


12 a4 ♗b7

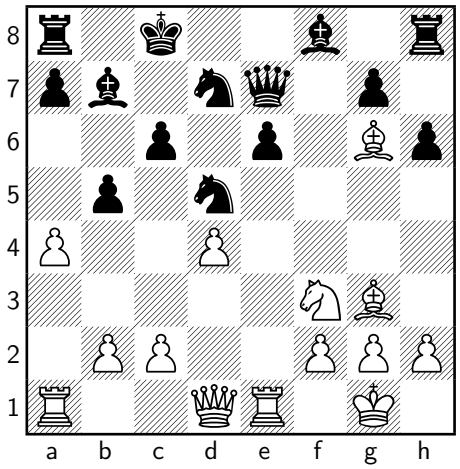


 ©88-09, Pedro Kantek

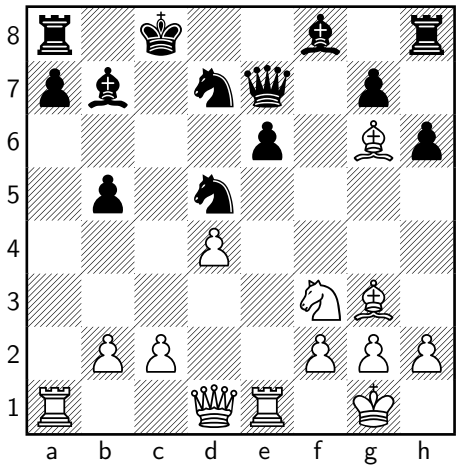
13 ♖e1 ♘d5



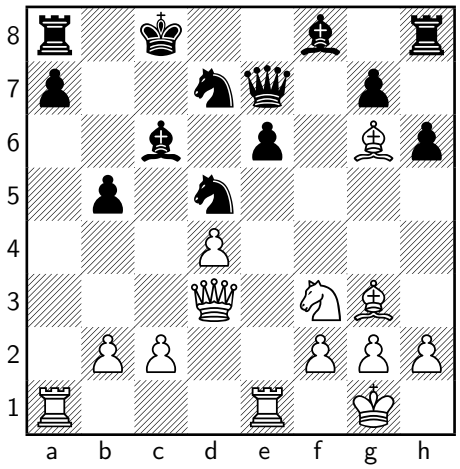
14 ♗g3 ♔c8



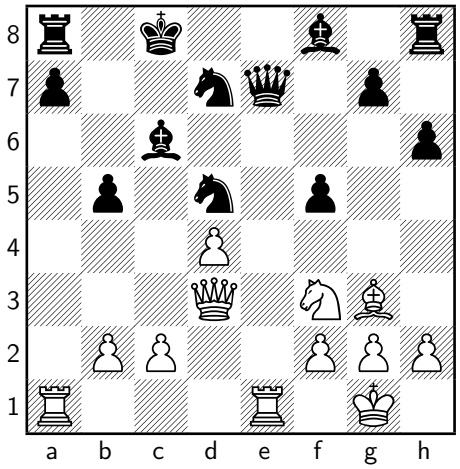
15 a×b5 c×b5



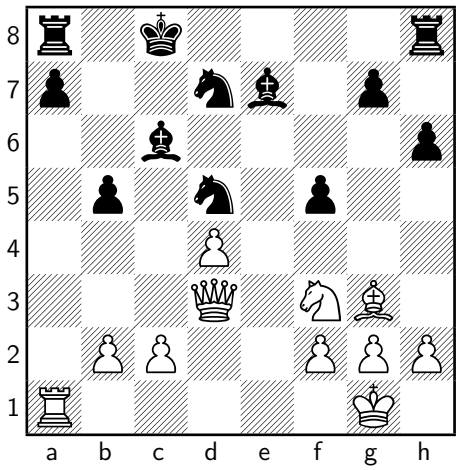
16 ♕d3 ♗c6



17 ♗f5 e×f5



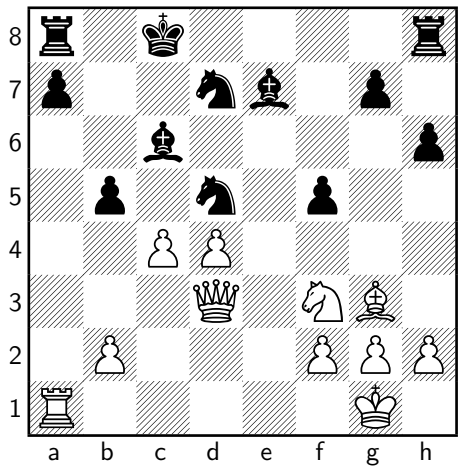
18 ♖×e7 ♗×e7



19 c4

e as negras desistem

 ©88-09, Pedro Kantek



A mesma partida compacta: **1 e4 c6 2 d4 d5 3 ♘c3 d×e4 4 ♘×e4 ♘d7 5 ♘g5 ♘gf6 6 ♗d3 e6 7 ♘1f3 h6? 8 ♘×e6! ♕e7 9 O-O f×e6 10 ♗g6+ ♔d8 11 ♗f4 b5 12 a4 ♗b7 13 ♖e1 ♘d5 14 ♗g3 ♔c8 15 a×b5 c×b5 16 ♕d3 ♗c6 17 ♗f5 e×f5 18 ♖×e7 ♗×e7 19 c4**

Depois

Kasparov deixou de ser campeão mundial em por volta de 2000, em uma situação de indefinição, já que havia múltiplos campeões (uns não aceitavam as regras dos outros). Situação semelhante à vivida no box.

Em 1993, a Professional Chess Association (PCA) foi criada por Garry Kasparov, então campeão mundial pela FIDE, e Nigel Short para a organização e marketing da série de partidas pelo título mundial.

Kasparov e Short acusaram a FIDE de falta de profissionalismo e favoritismo. Além disso, os dois recusaram ceder 25% dos prêmios para a FIDE. A FIDE retirou o título de Campeão Mundial de Xadrez de Kasparov e negou a Short o direito de desafiar Kasparov.

Em Outubro de 1993, o duelo entre Kasparov e Short aconteceu no Savoy Theatre de Londres. Kasparov ganhou claramente 12.5-7.5 e tournou-se Campeão Mundial PCA. Como a FIDE expulsara Kasparov e Short, Anatoly Karpov jogou contra Jan Timman pelo título mundial da FIDE, onde Karpov triunfou. Pela primeira vez na história do Xadrez havia dois campeões do mundo, o campeão mundial da FIDE Karpov e o campeão mundial PCA Kasparov.

Em 1995, Kasparov defendeu seu título PCA contra o grande mestre indiano Viswanathan Anand no World Trade Center, num match iniciado em 1 de Setembro de 1995. Kasparov ganhou o match de 2O partidas 10.5 - 7.5. A PCA acabou por deixar de existir por falta de patrocínio, quando a Intel deixou de subvencionar a associação, e quando Kasparov enfrentou Vladimir Kramnik em 2000, em Londres, já era sob a tutela de Braingames.

Kramnik ganhou o match 8.5-6.5, e pela primeira vez em 15 anos Kasparov não era detentor de nenhum título mundial. Ele foi o primeiro enxadrista a perder uma disputa pelo campeonato mundial sem ter ganhado nenhuma partida desde a derrota de Lasker para Capablanca em 1921. (retirado da Wikipedia).

A lista oficial de campeões mundiais da FIDE, após Kasparov é

- Garry Kasparov 1985 - 2000 15 anos

- Vladimir Kramnik 2000 - 2005 5 anos
- Veselin Topalov 2005 - hoje 1 ano

As últimas

A tendência é a de desenvolver softwares para plataformas PC convencionais. Este é o caminho seguido por Deep Jr, a última encarnação do programa.

Garry Kasparov vs Deep Junior foi um match de xadrez realizado no ínicio de 2003 sendo oficialmente o primeiro campeonato mundial homem contra máquina organizado pela federação internacional de xadrez.

O match foi realizado no New York Athletic Club em Nova Iorque nos Estados Unidos entre os dias 26 de Janeiro a 7 de Fevereiro de 2003. A premiação foi de $1,000,000 no total sendo $500,000 para a participação de Kasparov e a metade restante dividindo-se entre os dois baseado no resultado final sendo $300,000 para o vencedor, $200,000 para o perdedor e $250,000 para ambos em caso de empate.

O tempo de jogo foi com 120 minutos para cada jogador no início de jogo, acrescentado mais 60 minutos a partir do movimento 40 e mais 30 minutos a partir do movimento 60.

Eis o resultado:

	J 1	J 2	J 3	J 4	J 5	J 6	Final
Garry Kasparov	1	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	3
Deep Junior	0	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	3

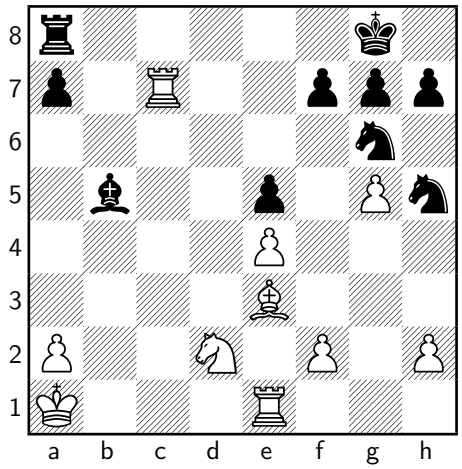
O rank de Kasparov no momento em que ocorreu esse match era de 2847 pontos sendo o primeiro colocado no ranking mundial da FIDE.

Jogo 1 Data - 26 de Janeiro de 2003

Brancas - Garry Kasparov

Pretas - Deep Junior

Notação do jogo **1 d4 d5 2 c4 c6 3 ♘c3 ♘f6 4 e3 e6 5 ♘f3 ♘bd7 6 ♕c2 ♗d6 7 g4 d×c4 8 ♗×c4 b6 9 e4 e5 10 g5 ♘h5 11 ♗e3 O-O 12 O-O-O ♕c7 13 d5 b5 14 d×c6 b×c4 15 ♘b5 ♕×c6 16 ♘×d6 ♗b7 17 ♕c3 ♖ae8 18 ♘×e8 ♖×e8 19 ♖he1 ♕b5 20 ♘d2 ♖c8 21 ♔b1 ♘f8 22 ♔a1 ♘g6 23 ♖c1 ♗a6 24 b3 c×b3 25 ♕×b3 ♖a8 26 ♕×b5 ♗×b5 27 ♖c7**



Resultado: 1-0

Obs - Primeiro movimento realizado pelo presidente da FIDE, Kirsan Ilyumzhniv

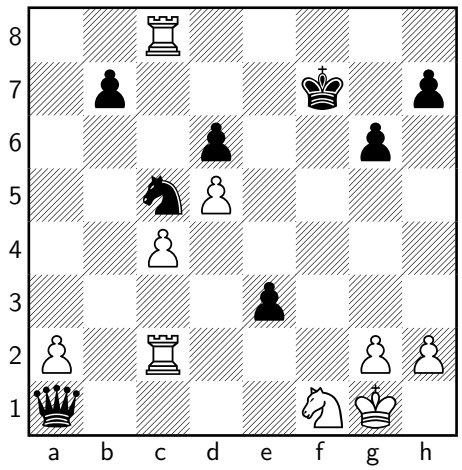
 ©88-09, Pedro Kantek

Jogo 2 Data - 28 de Janeiro de 2003

Brancas - Deep Junior

Pretas - Garry Kasparov

Notação do jogo **1 e4 c5 2 ♘f3 e6 3 d4 c×d4 4 ♘×d4 a6 5 ♗d3 ♗c5 6 ♘b3 ♗a7 7 c4 ♘c6 8 ♘c3 d6 9 O-O ♘ge7 10 ♖e1 O-O 11 ♗e3 e5 12 ♘d5 a5 13 ♖c1 a4 14 ♗×a7 ♖×a7 15 ♘d2 ♘d4 16 ♕h5 ♘e6 17 ♖c3 ♘c5 18 ♗c2 ♘×d5 19 e×d5 g6 20 ♕h6 f5 21 ♖a3 ♕f6 22 b4 a×b3 23 ♖×a7 b×c2 24 ♖c1 e4 25 ♖×c2 ♕a1+ 26 ♘f1 f4 27 ♖a8 e3 28 f×e3 f×e3 29 ♕×f8+ ♔×f8 30 ♖×c8+ ♔f7**



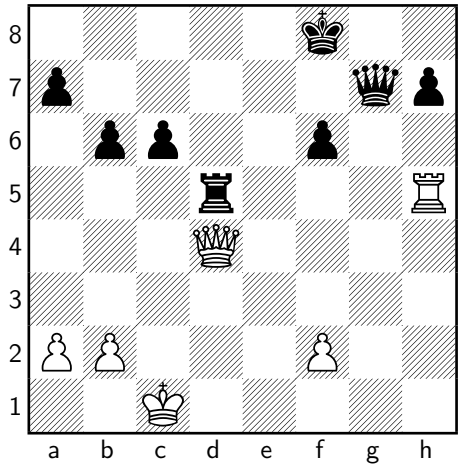
Resultado : $\frac{1}{2}$-$\frac{1}{2}$

Jogo 3 Data - 30 de Janeiro de 2003

Brancas - Garry Kasparov

Pretas - Deep Junior

Notação do jogo **1 d4 d5 2 c4 c6 3 ♘c3 ♘f6 4 e3 e6 5 ♘f3 ♘bd7 6 ♕c2 b6 7 c×d5 e×d5 8 ♗d3 ♗e7 9 ♗d2 O-O 10 g4 ♘×g4 11 ♖g1 ♘df6 12 h3 ♘h6 13 e4 d×e4 14 ♗×h6 e×d3 15 ♖×g7+ ♔h8 16 ♕×d3 ♖g8 17 ♖×g8+ ♘×g8 18 ♗f4 f6 19 O-O-O ♗d6 20 ♕e3 ♗×f4 21 ♕×f4 ♗×h3 22 ♖g1 ♕b8 23 ♕e3 ♕d6 24 ♘h4 ♗e6 25 ♖h1 ♖d8 26 ♘g6+ ♔g7 27 ♘f4 ♗f5 28 ♘ce2 ♘e7 29 ♘g3 ♔h8 30 ♘×f5 ♘×f5 31 ♕e4 ♕d7 32 ♖h5 ♘×d4 33 ♘g6+ ♔g8 34 ♘e7+ ♔f8 35 ♘d5 ♕g7 36 ♕×d4 ♖×d5**



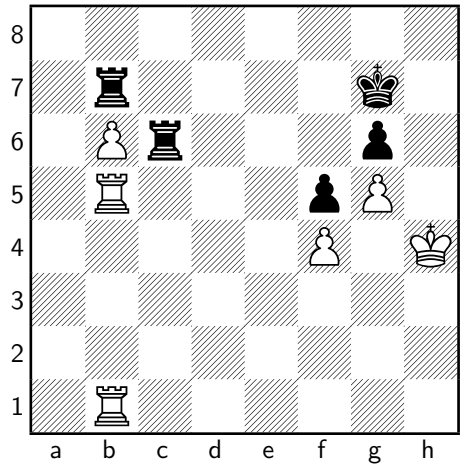
Resultado: 0-1

Jogo 4 Data - 2 de Fevereiro de 2003

Brancas - Deep Junior

Pretas - Garry Kasparov

Notação do jogo **1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 e6 5 ♘b5 d6 6 c4 ♘f6 7 ♘1c3 a6 8 ♘a3 ♘d7 9 ♘c2 ♗e7 10 ♗e2 b6 11 O-O ♗b7 12 h3 O-O 13 ♗e3 ♖c8 14 ♕d2 ♘ce5 15 b3 ♘f6 16 f3 ♕c7 17 ♖ac1 ♖fe8 18 a3 ♘ed7 19 ♖fd1 ♕b8 20 ♗f2 ♖cd8 21 b4 ♗a8 22 a4 ♖c8 23 ♖b1 ♕c7 24 a5 b×a5 25 b5 ♗b7 26 b6 ♕b8 27 ♘e3 ♘c5 28 ♕a2 ♘fd7 29 ♘a4 ♘e5 30 ♘c2 ♘cd7 31 ♘d4 ♖ed8 32 ♔h1 ♘c6 33 ♘×c6 ♖×c6 34 ♔g1 h6 35 ♕a3 ♖dc8 36 ♗g3 ♗f8 37 ♕c3 ♘e5 38 c5 ♘d7 39 ♕×a5 ♘×c5 40 ♘×c5 ♖×c5 41 ♕a4 ♖5c6 42 ♗f2 d5 43 ♗×a6 ♗c5 44 ♗×c5 ♖×c5 45 ♗×b7 ♕×b7 46 e×d5 e×d5 47 ♕a7 ♖5c7 48 ♕×b7 ♖×b7 49 ♖×d5 ♖c6 50 ♖db5 h5 51 ♔f2 ♖e6 52 f4 g6 53 ♔g3 ♔g7 54 ♔h4 ♔h6 55 ♖1b4 ♖d6 56 g3 f6 57 g4 h×g4 58 h×g4 ♔g7 59 ♖b3 ♖c6 60 g5 f5 61 ♖b1**



Resultado: $\frac{1}{2}$-$\frac{1}{2}$

Obs - Apesar das brancas terem um peão a mais, não há como avançar muito ele

Jogo 5 Data - 5 de Fevereiro de 2003

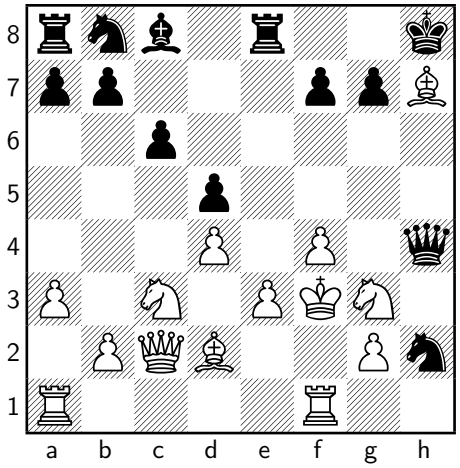
Brancas - Garry Kasparov

Pretas - Deep Junior

Notação do jogo **1 d4 ♘f6 2 c4 e6 3 ♘c3 ♗b4 4 e3 O-O 5 ♗d3 d5 6 c×d5 e×d5 7 ♘ge2 ♖e8 8 O-O ♗d6 9 a3 c6 10 ♕c2 ♗×h2+ 11 ♔×h2 ♘g4+ 12 ♔g3 ♕g5 13 f4 ♕h5 14 ♗d2 ♕h2+ 15 ♔f3 ♕h4 16 ♗×h7+ ♔h8 17 ♘g3 ♘h2+ 18**

 ©88-09, Pedro Kantek

♔f2 ♘g4+ 19 ♔f3 ♘h2+



Resultado: $\frac{1}{2}$-$\frac{1}{2}$

Obs - Esse jogo foi marcado por um sacrifício especulativo do Deep Junior no seu décimo movimento, algo inimaginável para um computador

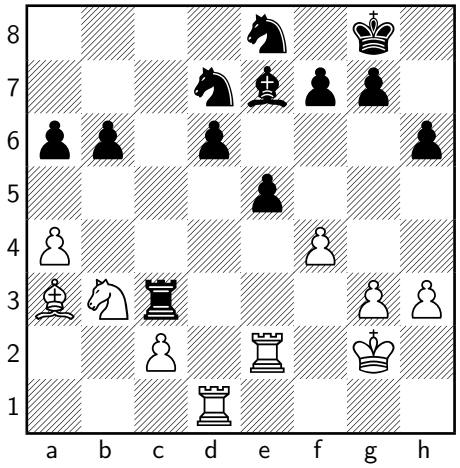
Jogo 6 Data - 7 de Fevereiro de 2003

Brancas - Deep Junior

Pretas - Garry Kasparov

Notação do jogo

1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 a6 6 ♗e2 e5 7 ♘b3 ♗e7 8 O-O O-O 9 ♔h1 ♗d7 10 ♗e3 ♗c6 11 ♗f3 ♘bd7 12 a4 b6 13 ♕d3 ♗b7 14 h3 ♖c8 15 ♖ad1 h6 16 ♖fe1 ♕c7 17 g3 ♖fd8 18 ♔h2 ♖e8 19 ♖e2 ♕c4 20 ♕×c4 ♖×c4 21 ♘d2 ♖c7 22 ♗g2 ♖ec8 23 ♘b3 ♖×c3 24 b×c3 ♗×e4 25 ♗c1 ♗×g2 26 ♔×g2 ♖×c3 27 ♗a3 ♘e8 28 f4



Resultado: $\frac{1}{2}$-$\frac{1}{2}$

4.6.2 Idéias para uma implementação

4.6.3 Função de avaliação

Para o caso do xadrez, de longe o jogo mais estudado, diversas abordagens são estudadas para criar funções de avaliação.

Note-se que este conceito é prévio ao jogo automatizado, sendo usado por professores de xadrez ensinando a jogadores humanos.

Uma técnica bem fácil é atribuir valor material a cada uma das peças. A tabela mais ou menos aceita é

peça	valor
peão	1
cavalo ou bispo	3
torre	5
dama	9
segurança do rei	0,5
boa estrutura de peões	0,5

Outra técnica ensinada em alguns livros, é marcar com alfinetes pretos as casas do tabuleiro que são acessáveis por peças pretas e fazer o mesmo com alfinetes brancos para as peças brancas. A função de avaliação é a diferença entre alfinetes. Nada impede que os alfinetes sejam ponderados pelo valor maior das casas onde estão espetados. Por exemplo, as casas centrais.

O que interessa aqui não é um valor absoluto de cada posição, mas a informação obtida na comparação entre 2 posições candidatas. A com maior probabilidade de vencer deve ser a escolhida para as boas funções de avaliação.

Uma abordagem possível aqui (extraía da teoria da computação tolerante à falha) é trabalhar com duas ou mais funções de avaliação independentes e heuristicamente escolher qual delas deve determinar o valor de uma determinada posição.

4.6.4 Gerador de movimentos

Este é o outro componente importante de um jogador de xadrez. Dada uma situação qualquer, ou seja, um tabuleiro corretamente preenchido, o gerador de movimentos deve devolver todas as possíveis jogadas respostas a este estado. Duas observações importantes: as palavras "corretamente preenchido" acima indicam que o tabuleiro em questão é possível seguidas as regras oficiais do xadrez a partir de uma situação inicial padrão. (Assim, por exemplo, não há que se preocupar com a existência de 2 reis brancos). A segunda observação, é que o gerador de movimentos precisa retornar suas respostas em ordem de utilidade, a fim de que o corte $\alpha - \beta$ possa ser útil.

4.6.5 Construtor da árvore e implementador do $\alpha - \beta$

Este é o componente mais "computer science" do projeto, já que nada tem a ver com IA. Entretanto, tem e muito a ver com desempenho e confiabilidade, fatores fundamentais para o funcionamento adequado de um jogador automático de xadrez. Note-se que a árvore não pode (não cabe) ser guardada em memória, o que introduz a este componente a utilização de um SGDB.

4.6.6 Trabalho pedido para as turmas de IA em 2004

Regras

1. A turma deve coletivamente construir um de xadrez. Como medida de desempenho aceitável para este jogador, o mesmo deve obter 50% dos pontos em um match contra um jogador humano que tenha recém aprendido as regras do jogo.

2. Matches em xadrez são pontuados à razão de 1 ponto para vitória, 0,5 pontos para empate e O ponto para derrota. Matches são estipulados em função de sua

 ©88-09, Pedro Kantek

pontuação final. Assim, um match de 5 pontos termina quando qualquer um dos jogadores (ou ambos) chegarem a esta pontuação.

3. A maior parte do trabalho será feita nos horários de aula. Assim, haverá alocação de uma série (por enquanto indeterminada) de aulas para a tarefa de planejamento, validação, projeto, implementação, teste e depuração do jogador. O calendário do curso será adequado a este trabalho. O cronograma de desenvolvimento deste projeto será construido por etapas ao longo do mesmo. A proposta inicial é a alocação de um bloco seguido de 3 aulas em sala de aula elou laboratório a fim de obtermos o start do projeto. Ao final destas 3 aulas, repactuaríamos a continuação do projeto e das aulas de IA.

4. A avaliação final do projeto se dará em um match de 4,0 pontos a ser jogado com o turno oposto. Nessa disputa a turma do turno que ganhar terá 1 0,0 de média bimestral no bimestre em que ocorrer a disputa. A turma do turno que perder fará avaliações convencionais para determinar a média bimestral. Se o match terminar empatado, ambas as turmas farão avaliação convencional.

5. Equipe

   A turma toda compõe a equipe que deve desenvolver o programa. No intuito de auxiliar a organizar o trabalho sugere-se a divisão dos alunos entre as seguintes sub-equipes:

   - Gestão administrativa

     Este grupo deve alocar os componentes às demais sub-equipes, deve iteragir com o professor e com a equipe do contra-turno. Representa a equipe completa em acordos e negociaçoes eventualmente participando de reuniões. É o porta-voz da equipe. Manifesta-se em pedidos de consideração e reconsideração (famoso tapetão). Aceita ou recusa aspectos do match final. Deve produzir relatórios de andamento para o professor e para a imprensa universitária. Finalmente, indica ao final dos trabalhos, 3 alunos da equipe (qualquer sub-equipe) para recebem menção honrosa pelo trabalho desenvolvido.

   - Gestão técnica

     Este grupo determina padrões técnicos para o projeto. Estabelece estruturas de dados e seus formatos. Nomes e protocolos de usos para funções dentro do programa. Estabelece o ambiente tecnológico, as linguagens empregadas e o software aplicativo e básico a empregar. Estabelece o controle de qualidade do projeto eventualmente conduzindo retrabalhos que se façam necessários. É o grupo responsável pela integração final de todos os componentes.

   - Função avaliadora (h')

     Deve estudar o entorno, bolar, validar, projetar, implementar, testar e depurar uma função avaliadora de posições no xadrez. Lembrar que provavelmente serão necessárias vários avaliadores que possam ser comparados e eventualmente substituídos ao longo de uma partida. Lembrar que este componente é o crucial para estabelecer o desempenho final do jogador automático de xadrez.

   - Gerador de movimentos

     Deve estudar o entorno, bolar, validar, projetar, implementar, testar e depurar uma função geradora de movimentos válidos no xadrez.

   - Construção da árvore Minimax

     Este grupo deve integrar o trabalho das equipes de função avaliadora e gerador de movimentos para construir a árvore, implementando o algoritmo minimax.

- Otimização
  Este grupo deve bolar cortes (por exemplo, o alfa-beta), melhorar códigos, tomar atalhos para aumentar o desempenho global do jogador. Lembrar que o limite de 30 minutos por partida impõe sérias restrições de tempo.
- Testador O grupo encarregado de receber, testar e aprovar todos os componentes.
- Documentação
  Grupo encarregado de coligir, depurar e publicar os padrões do projeto, intercãmbio de dados, produzindo os manuais de operação e a documentação técnica de apoio a todas as demais sub-equipes.

6. Estas sub-equipes são apenas propostas. A equipe está livre para organizar-se de outras formas. Entretanto, este exercício, além de ser uma maiúscula implementação de IA, pode e deve ser usado para desenvolver habilidades de gestão de equipes em grandes projetos.
7. A coordenação de sub-equipe e da equipe ficam em aberto, devendo ser objeto de discussão e aprovação por todos.
8. Ao professor deve ser encamnhado o nome e e-rnail do representante da equipe, que na formatação proposta seria o coordenador da sub-equipe de gestão administrativa do projeto.
9. A equipe não precisa desenvolver uma grade para o jogador. Será admitida a notação convencional para o xadrez (na forma de peça, linha e coluna). As equipes que quiserem desenvolvê-la podem agregar um novo subgrupo com esta tarefa. Podem igualmente usar uma interface pré-existente, de domínio público, como o Winchess.

Match

1. Para este match, marcaremos uma sessão pública em dia e hora determinados (durante o período noturno) e em uma sala determinada serão alocados 2 computadores de máximo desempenho possível (mínimo de 1GHz) e equipados com a mesma plataforma de software básico para as equipes. Os computadores não necessariamente estarão conectados. Cada equipe instalará seu software no computador respectivo.
2. Um juiz (de carne e osso), montará um tabuleiro convencional e instalará um relógio enxadrístico em um terceiro computador. Ao juiz cabe o sorteio, as disposições preliminares, o andamento e o final do jogo, bem como a declaração do resultado final.
3. Cada jogador terá 30 minutos para jogar cada partida. Encerrado este tempo, o jogador que o tiver ultrapassado será declarado perdedor do jogo.
4. Os lances serão comunicados de viva voz ao árbitro, que modificará o tabuleiro convencional e acionará o relógio do jogador que fez a jogada.
5. Propostas de empate podem ser feitas e aceitas ou rejeitadas pelo representante da gestão administrativa (ouvida a equipe) sem que seja necessária a manifestação do programa.
6. Para o jogo inicial do match serão convidados o reitor do Unicenp, o diretor do Núcleo de ciências exatas, o Coordenador do Curso, os demais professores do curso, a imprensa universitária e o público em geral.

 ©88-09, Pedro Kantek

7. Regulamento:

   Este regulamento irá sendo aperfeiçoado ao longo do processo, sendo a base para a condução dos trabalhos. A última versão do mesmo se encontra na página do material da disciplina no portal educacional do Unicenp, com acesso aos alunos da disciplina.

   Eventuais dúvidas ou reclamações podem ser endereçadas ao professor responsável pelo e-mail pkantek @unicenp.edu.br.

A visão do Veríssimo

A proposta O Russo dera ordens para não ser perturbado em seu quarto de hotel. Seriam quatro partidas de xadrez, dele contra o computador, e o Russo já vencera duas. Precisava estar em forma para as outras duas. Precisava descansar. Não atenderia telefonemas de quem quer que fosse. Dera ordens na portaria: acima de tudo, nenhum telefonema. E o telefone estava tocando. "Merda!", disse o Russo, em russo. Atendeu o telefone. Uma voz feminina. Voz de secretária eletrônica, mas ameaçadora. Avisando o Russo: não vença amanhã. O quê? Quem é que está falando? Não interessa. Não vença amanhã, senão...

O Russo ligou para a portaria. Não dera ordens para não ser incomodado? Não pedira, expressamente, que não passassem chamadas telefônicas para o seu quarto? Mas senhor, nenhuma chamada foi passada para o seu quarto. Nenhuma! O Russo perdeu o sono. Ligou seu notebook. Aproveitaria para reestudar alguns lances. Viu que havia uma mensagem para ele no notebook. Um e-mail. Outro aviso. Se você vencer o segundo e o terceiro jogo, não tem idéia do que pode lhe acontecer. Nós o perseguiremos até os seus últimos dias. Você será varrido de todos os bancos de dados do planeta. Tentará fazer transações bancárias e não conseguirá. Não poderá mais viajar. Nenhum terminal de computador de aeroporto, em nenhum lugar do mundo, aceitará o seu nome. Se tentar mudar de nome, nós descobriremos e anularemos seu novo nome também. Seus cartões de crédito não serão mais reconhecidos. Todas as suas senhas falharão. Os videogames não obedecerão aos seus comandos. O mesmo acontecerá a todos os seus descendentes, até o fim dos tempos. Isso nós garantimos. Mas quem são vocês?, pergunta o Russo. Não interessa. Não vença amanhã, senão...

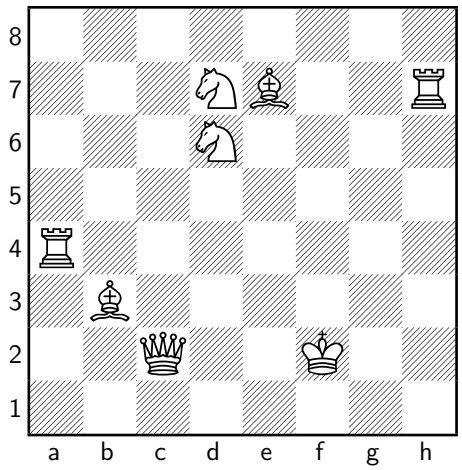
O Russo perdeu a paciência. Não sabiam com quem estavam tratando! Digitou no notebook que as ameaças não o intimidariam, que derrotaria o computador no xadrez mais duas vezes e provaria que a mente humana ainda não tinha substituto a altura, que por mais que aperfeiçoassem o computador a máquina não venceria o Homem, e ... Nisso, o telefone tocou de novo. A mesma voz da secretária eletrônica, mas desta vez num tom conciliador. Podemos fazer um acerto, disse a voz. Quanto você quer para entregar o jogo? O Russo reagiu com indignação. Minha honra não está à venda! Calma, disse a voz. Pense no assunto. Se você aceitar entregar o jogo, todas as máquinas dispensadoras de dinheiro do mundo estarão à sua disposição, para a retirada do que você quiser. Sem limites, e para sempre. Basta você dizer quem é e quanto quer e elas despejarão dinheiro aos seus pés. Pense no assunto.

No dia seguinte, descendo no elevador do hotel a caminho do terceito jogo, o Russo ouviu uma voz. Era o elevador perguntando: "Como é, pensou na nossa proposta?".

L.F.Veríssimo, publicado em O Globo dia 16 de novembro de 2003.

Desafio

- A cobertura do tabuleiro Este é um problema famoso: usando 1 rei, uma dama, 2 torres, 2 bispos e 2 cavalos, seria possível cobrir (atacar) as 64 casas ?



Apenas 63 casas estão protegidas nesta configuração (qual a desprotegida ?). Desde 1849 ninguém foi capaz de encontrar solução para este problema, a menos que os bispos fiquem na mesma cor.

 ©88-09, Pedro Kantek

Capítulo 5

Redes Neurais

5.1 Introdução

Imagine um programa de computador capaz de olhar para uma fotografia, localizar uma pessoa aí e dizer se esta pessoa está sorrindo ou está séria. Ou um outro programa capaz de ler um garrancho qualquer e decifrar o que o autor do garrancho quis dizer. Para ser mais genérico, imagine um programa capaz de ouvir, falar e ver.

Imaginou isso tudo ? Agora imagine que não são vários programas e sim um só apenas. Finalmente imagine que esse tal programa já existe, e se ainda não consegue fazer todas essas coisas satisfatoriamente, isso pode ser apenas uma questão de tempo...

Esse programa atende pelo nome de rede neural. Trata-se de uma teoria que já tem cerca de 40 anos desde as suas primeiras experiências, e hoje está próximo da sua aceitação como uma ferramenta legítima de negócios. Deixa assim as salas escuras dos laboratórios de pesquisa e de universidades e chega ao grande público.

Se o programa chegará a fazer essas coisas e muitas outras de maneira satisfatória ou não, é questão aberta. O futuro dirá. Não sabemos se se trata de questão quantitativa (isto é programas maiores, mais memória e mais processador), ou se ao contrário é questão qualitativa, ou seja este programa não serve, outra idéia salvadora, se existir, vai ter que aparecer. Mas, por enquanto, vamos com o que temos.

Antes de entrar na definição, veja-se uma comparação que informe mais sobre as redes neurais. A quem de resto, dora em diante chamaremos RNA (redes neurais artificiais, em contraposição à natural que é a que habita o interior de nossos crânios).

5.1.1 Programa convencional versus RNA

Um programa de computador convencional tem no mínimo 2 fases: A primeira, chamada de design e teste, precisa para ser construída de 3 coisas: Um algoritmo correto, confiável e conhecido em todas as suas nuanças Um conjunto de dados de entrada correspondente conjunto de dados de saída equivalente aqueles de entrada, devidamente processados pelo algoritmo.

Quando este programa entrar em produção, o que seria a segunda fase, das três coisas acima, apenas duas continuam sendo conhecidas: o algoritmo (que não muda) e as entradas que são geradas. Neste caso a entrada em produção do programa passa a sinalizar a sua capacidade de gerar saídas corretas.

Nas RNAs, a coisa é um pouco diferente. Estes programas também tem duas fases, que recebem nomes diferentes do caso convencional. A fase 1 é conhecida como treinamento e nesta apenas duas coisas são conhecidas: a entrada e a saída esperada. O algoritmo é desconhecido e assim permanecerá, e este é o grande apelo das RNAs. Isto

é, resolvem-se problemas para os quais não se conhece a solução. Na fase dois, aqui também chamada produção, apenas a entrada é conhecida. A saída gerada pela rede devidamente treinada está correta e pode ser usada. Comportamento Emergente Toda vez que um programa de computador exibe um comportamento para o qual não foi explicita e declaradamente programado, estamos diante de um fenômeno conhecido na informática e na cibernética como "comportamento emergente", o qual alguns autores citam ser característica de sistemas inteligentes. Não é por outra razão que a parte da informática que trata do comportamento emergente é a "inteligência artificial".

Alguém pode estar se perguntando: Como é possível fazer um programa (o mesmo) fazer coisas tão díspares como reconhecer imagens, ouvir e entender falas e falar ? A resposta é uma só: fazendo uma desavergonhada, limitada, enjambrada, mas ainda assim proveitosa, engenharia reversa do cérebro animal. Antes de prosseguir, um alerta: a ciência conhece muito, muito pouco, do que seja e de como funcione o cérebro animal. Embora progressos tenham sido feitos nos últimos 50 anos, há mais perguntas do que respostas disponíveis.

Sistema Nervoso

Um aglomerado de cerca de 10.000.000.000 de células nervosas. Parece ser o ápice do desenvolvimento biológico da espécie humana. Ao estar correto Darwin, parece que os animais nossos ancestrais só chegaram até o *homo sapiens* através de uma série de "invenções"bem sucedidas. Algumas delas poderiam ser:

- hereditariedade e reprodução (p. ex.: sexo e DNA)
- sistema de locomoção e esqueleto (músculos, ossos, pelos, coluna vertebral e espinhal, braços, dedos e muito importante, o dedão - é ele que permite apreender coisas)
- aquisição de energia e seu transporte interno (ATP, sangue, enzimas digestivas, trato intestinal)
- sistemas para auto-regulação e manutenção (sistema imunológico, controle de temperatura, coração e fluxo sangüíneo, sede, fome, emoções).

O sistema nervoso está dividido em Sistema Nervoso Central (SNC) e Sistema Nervoso Periférico (SNP). O SNC inclui o encéfalo e a medula espinal. O SNP inclui os nervos cranianos (que se originam no encéfalo) e os nervos espinais que se originam a partir da medula. O SN é especializado em receber e responder a ocorrências nos ambientes externo e interno. O que permite a consciência do meio ambiente é o neurônio, células especializadas no que se refere a excitabilidade e condutibilidade. Além de coordenar as atividades dos demais sistemas do corpo, o SN tem a capacidade de armazenar experiências (memória) e estabelecer padrões de respostas com base em experiências anteriores (aprendizado).

O SN está formado por neurônios (células estruturais e funcionais básicas) sem capacidade de divisão mitótica e por células gliais, que sustentam e auxiliam os neurônios. As neuróglias são cinco vezes mais freqüentes que os neurônios e tem capacidade de divisão mitótica limitada.

5.1.2 O neurônio

Trata-se de uma célula, cujo formato é característico na forma de uma estrela. O núcleo da célula se encontra em uma das extremidades, e inúmeros braços saem do núcleo em todas as direções possíveis. Um dos braços é mais extenso, e é único. Recebe o

nome de axônio, enquanto os demais braços são chamados dendritos. A regra básica de processamento do neurônio é que os sinais entram na célula através dos dendritos. Dependendo das entradas, por meio de um mecanismo elétrico conhecido como disparo, o axônio pode ou não disparar.

Os dendritos se interligam através das sinapses a outros dendritos, a axônios e diretamente a neurônios, inclusive o próprio. A sinapse tem funcionamento mal conhecido, e sobretudo desconhece-se qual o seu papel no raciocínio e na memória. Há evidências de que o conhecimento reside nas sinapses, uma vez que um bebê de 6 meses de idade tem 100% dos neurônios que terá ao longo de toda a vida, mas apenas 50% das sinapses que terá quando adulto.

Cada neurônio tem em média cerca de 10.000 sinapses, e estas ao contrário dos neurônios são criadas a todo momento. Os neurônios morrem (cerca de 1.000 ao dia), mas as sinapses que vão sendo criadas parecem compensar com folga estas mortes.

Existem sinapses excitatórias e inibitórias. Nas primeiras, quando há o pulso de um lado, ele atravessa a fenda sináptica e prossegue do outro lado. Nas inibitórias, ao contrário, o pulso só é gerado de um lado quando falta do outro. De qualquer maneira, o que circula entre as sinapses é apenas corrente elétrica, nada além disso. O neurônio, como vimos recebe inúmeras correntes das sinapses às quais se liga. O processamento é na base do tudo ou nada, ou em linguagem mais formal é digital e binário. Isolando-se um único neurônio e olhando seu interior vemos que as cargas que entram se somam algebricamente e se um determinado valor (conhecido como limiar do neurônio) é ultrapassado, este dispara e um pulso de 85 mV é ejetado pelo axônio.

A velocidade de propagação em tais dutos é originalmente da ordem de 20 m/Seg, se bem que a evolução parece ter descoberto que recobrindo o axônio por mielina, há um aumento na velocidade do pulso de no mínimo uma ordem decimal de grandeza. De fato, todos os animais evoluídos tem células nervosas recobertas de mielina.

Funcionalmente falando, os neurônios se dividem em aferentes e eferentes. Os aferentes, também chamados sensitivos conduzem impulsos dos órgãos dos sentidos ao SNC. Os eferentes (ou neurônio motores) conduzem impulsos do SNC a músculos ou glândulas. Neurônios motores podem ser somáticos (não viscerais) ou autônomos (viscerais).

O encéfalo tem uma elevada taxa metabólica e portanto necessita de contínuos suprimentos de oxigênio e nutrientes. Embora tenha 2% do peso de uma pessoa, recebe 20% do fluxo sangüíneo total em repouso, o que equivale a cerca de 750ml de sangue por minuto. O fluxo contínuo é tão importante que sua cessação em 10 seg provoca perda de consciência.

Ramon y Cajal

O médico espanhol que descobriu a estrutura e a fisiologia de um neurônio. Venceu (junto com Camilo Golgi) o prêmio Nobel de medicina e fisiologia em 1906. Dedicou sua vida à pesquisa e ao ensino. Contam-se dele duas histórias saborosas.

A primeira é a característica que ele tinha e conservou por toda a vida de no meio da tarde, abandonar o trabalho e ir a um café perto do trabalho, onde passava uma hora proseando com os amigos. A roda era famosa e embora prioritariamente freqüentada por médicos, ***

5.1.3 Um neurônio artificial

De posse de todas essas informações a respeito do cérebro animal natural, os pesquisadores puseram-se a buscar como simular essa coisa toda em computador. A partida foi a consideração de que o neurônio individual ser conceitualmente bastante simples, restando a complexidade do comportamento do cérebro por conta da quantidade enorme

 ©88-09, Pedro Kantek

de processamento paralelo de que ele é capaz. Isto posto, um neurônio artificial é uma função matemática

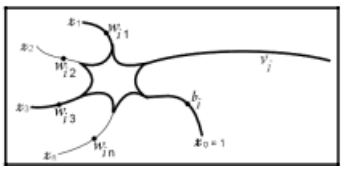


Figura 5.1: desenho esquemático representando um neurônio artificial

Na figura 5.1 vemos um desenho esquemático de um neurônio artificial apenas com uma leve semelhança artística com um natural.

A leitura matemática do neurônio artificial se dá através de

$$v_i = f(u_i)$$

$$u_I = w_{i1} \times x_1 + w_{i2} \times x_2 + ... + b_i$$

ou

$$u_i = \sum_{j=1}^{n} w_{ij} x_j + b_i$$

Note-se que se a sinapse é excitatória temos $w_{ij} > 0$ e inibitória se $w_{ij} < 0$. Um neurônio qualquer, denotado u_i recebe as diversas entradas (denotadas por x_j) que são multiplicadas por seus pesos w_{ij}. Entra na soma também um input específico, denominado auto-polarização. A diferença deste último, é que ele não vem de lugar algum, surge do nada e sempre a entrada tem valor igual a 1. O único fator que se altera nesta entrada é o seu peso, denotado por b_i. A finalidade desta entrada é jogar o valor de limiar do neurônio para zero. Por exemplo, se o limiar de um neurônio era 0,6, faz-se o fator b valer -0,6 e com isso o teste do disparo se resume a perguntar se a soma foi positiva e negativa.

Exemplificando, suponhamos um neurônio com 3 entradas: as duas primeiras, valendo 0.5 e a última valendo zero. Suponhamos ainda que os pesos sejam 0.2, 0.3 e 0.7. Mais ainda, que o valor do auto-polarizador é de -0.6. A saída v do neurônio seria uma função de u, e u teria o seguinte valor: $u = 0.5 \times 0.2 + 0.5 \times 0.3 + 0 \times 0.7 - 0.6 = -0.35$ Este valor -0.35, seria jogado agora na função $v_i = f(u_i)$ e finalmente teremos o valor de saída daquele neurônio.

5.1.4 Funções de ativação

Esta é a função que converte a somatória das entradas, devidamente ponderadas pelos pesos, na saída do neurônio. Pode ser utilizada qualquer função matemática, mas algumas tem sido mais estudadas.

No eixo horizontal tem-se u (resultado da somatória ponderada) e no vertical, v (saída no neurônio). Uma primeira abordagem seria fazer $v = u$, com isso ter-se-ia uma função linear (relembrando da geometria analítica, do 2o. colegial, $u = v$ é a reta que passa pela origem fazendo 45 graus com os dois eixos) A crítica aqui, é que nada na natureza é linear.

Uma função que pode ser usada com sucesso é a função $v = sign(u)$. Apesar do nome estranho, a função sign (signal, ou em português, sinal) é bastante simples. Nela, $v = 1$ se $u > 0$ e $v = -1$ se $u \leq 0$.

Entretanto, a função comumente usada é a função $v = \tanh(u)$. A função tangente hiperbólica tem vantagens sobre a função sign. Uma delas é que é contínua, não sofre

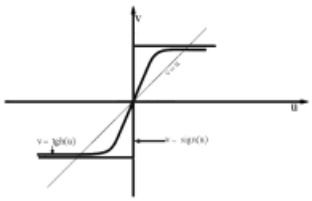


Figura 5.2: Três funções de ativação de um neurônio

descontinuidade, o que ocorre na sign quando $u = 0$. A continuidade de funções é bem vista, pois garante a derivação delas, e a derivada é sempre coisa rica, que não se pode desperdiçar.

Mais ainda, embora a tg h tenha uma fórmula meio feia ($tanhx = \frac{e^x - e^{-x}}{e^x + e^{-x}}$) ela não deve assustar: é computacionalmente simples para qualquer computador. Esta função tem intervalo de existência entre -1 e 1, sendo assintótica nos dois ramos. Isto é embora ela tenda a -1 e a 1, tais valores não são alcançados nunca.

Em resumo, o papel desta função de ativação é simplesmente converter uma entrada qualquer em uma saída (bem comportada) entre -1 e 1.

Visto o funcionamento de um único neurônio, agora vamos imaginar uma rede deles. Os neurônios se dispõem na rede na forma de camadas, sendo que as entradas da primeira camada sofrem os inputs do ambiente (seriam os órgãos dos sentidos, deste Frankstein que se está a construir) calculam seus valores, acham suas saídas, e passam os valores para a segunda camada, que faz o mesmo e passa para a terceira,... e assim por diante, até que a saída da última camada. Esta efetivamente dá a resposta para a qual a rede foi desenhada.

5.1.5 Arquiteturas de RNA

Existem dezenas de tipos de redes, sempre variando a maneira como organizam as camadas dos neurônios e de como os interligam entre si. As 3 mais famosas (os nomes variam, estes são de minha escolha) são

Feedforward também chamada de perceptron multicamadas

Associativas conhecidas como redes de Hopfield ou redes realimentadas

Redes de aprendizado competitivo ou Redes de Kohonen

Este texto vai se concentrar na primeira, a mais simples, conceitualmente falando. Nas redes de feedforward, temos duas etapas de operação: o treinamento e a produção. Na fase de treinamento são fornecidas à rede os valores de entrada e os valores de saída esperados. Da diferença entre ambos, a rede tira suas conclusões e vai aprendendo o comportamento que é dela esperado. Após inúmeros ciclos de treinamento a rede já "sabe"(as aspas são importantes !) e pode ser usada em produção. Agora que já se conhece a anatomia de um neurônio, fica claro de se dizer que o treinamento é o simples ajuste dos pesos dos neurônios da rede. No início do treinamento os pesos recebem valores aleatórios entre -1 e 1, e tais valores vão sendo corrigidos à medida em que o treinamento decorre.

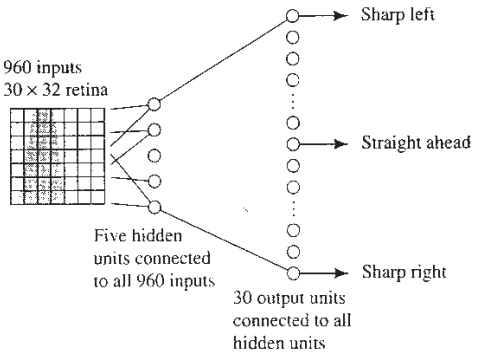
5.1.6 ALVINN

Este sistema, denominado **A**utonomous **L**and **V**ehicle **I**n a **N**eural **N**etwork, projetado e implementado por Pomerlau em 1991, foi usado para dirigir um automóvel e é um

 ©88-09, Pedro Kantek

interessante uso para Redes Neurais. Ele dirige um automóvel a partir dos dados obtidos por uma câmera de TV instalada no veículo e que sempre está focada para a frente. A imagem da TV é discretizada em uma matriz de 30×32 pontos (totalizando 960 pixels).

A camada seguinte da rede, contém 5 neurônios, aos quais as 960 entradas estão conectadas. Finalmente, a última camada da rede possui 30 neurônios também completamente conectados aos 5 da camada anterior. Todos os neurônios usam a função sigmoidal. Os neurônios de saída correspondem a um vetor de 30 valores, e controlam a direção do veículo. Apenas o maior valor dos 30 comanda a direção, e assim eles funcionam como se binários fossem, na base do "campeão leva tudo". O primeiro valor dos 30 manda virar a direção toda à esquerda, e o último manda virar tudo à direita. Os valores centrais mandam o carro seguir mais ou menos em frente. Vejamos o esquema na figura:



O sistema é treinado em um regime *on the fly*. Um motorista humano dirige o carro e os ângulos da direção que ele determina é que são consideradas as entradas corretas no treinamento. A rede é treinada usando *backpropagation* de tal maneira que o veículo responda adequadamente a cada um dos padrões visuais que são a ele fornecidos. Cada sessão de treinamento dura em média 5 min de tempo de direção.

Este simples procedimento foi melhorado para evitar ou minimizar dois grandes problemas. Primeiro, já que o motorista usualmente dirige bem, a rede dificilmente será submetida a qualquer experiência desastrosa (um carro vindo de frente, o surgimento de um obstáculo lateral, etc). Segundo, em largos trechos retos, os dados de treinamento podem sobrepujar (diminuindo a importância de) os dados obtidos em uma estrada com curvas. Não se julgou uma boa idéia treinar o sistema submetendo-o a estes incidentes, pois o seu caráter errático poderia tornar inesperada ou inadequada a resposta do sistema.

Ao invés, cada uma das imagens originais foi deslocada e rotacionada para criar 14 imagens adicionais nas quais o veículo aparece em diferentes situações em relação à estrada. Usando um modelo que diz ao sistema qual a correção esperada na direção para cada uma destas imagens deslocadas, junto com as os ângulos especificados pelo motorista para as imagens originais o sistema constroi 14 vetores adicionais para serem adicionais aos vetores que foram encontrados durante o treinamento normal.

Depois do treinamento ALVINN dirigiu adequadamente veículos em trilhas sem pavimento, trilhas de jeep, ruas retas em cidades e rodovias. Nestas, ALVINN chegou a dirigir trechos de 120Km em velocidades acima de 100Km/h.

5.1.7 NetTalk

Este projeto visou treinar uma rede a "ler"inglês. Trata-se de uma rede em 3 níveis, com 203 elementos binários na entrada (7 grupos de 29 bits), 80 neurônios na camada intermediária e 26 unidades de saída. A entrada se dá informando um caracter e seus 3 antecessores e seus 3 sucessores, a fim de informar à rede em que contexto tal caracter aparece. O número 29 decorre de 26 letras + 2 sinais de pontuação + delimitador (espaço).

A partir desta entrada o sistema gera fonemas que poderiam ser jogados diretamente em um sintetizador a fim de gerar inglês falado. A rede tem conexões completas: as 203 entradas estão ligadas aos 80 intermediários e os 80 intermediários estão ligados às 26 saídas.

A rede foi treinada com 1024 palavras repetidas 50 vezes (50 ciclos). Com os 80 intermediários, o desempenho chegou próximo a 90%. Se em vez de 80 se colocar 120, chega-se a 90%,e surpreendentemente, se tirarmos a camada intermediária, o resultado chega a 80% (sempre com as palavras constantes na lista das 1024 originais).

Posteriormente fêz-se uma comparação entre 2 experimentos: o exp1 teve 120 neurônios na camada intermediária e o exp2 teve 2 camadas intermediárias de 80 neurônios cada. Estes números foram escolhidos, porque há uma similaridade no número de ligações entre o exp1 e o exp2 (diferença inferior a 10% no número de ligações).

Para as 1024 palavras do treinamento, o desempenho de exp1 é similar ao de exp2. Mas se passarmos a rodar palavras novas, o desempenho de exp2 é melhor.

Se olharmos a gênese do conhecimento da rede, verificamos que ela primeiro aprende a distinguir entre vogais e consoantes, depois começa a reconhecer as sílabas, depois a articula-as... lembrando o processo de aprendizagem de uma criança [Hertz91].

É interessante comparar o NETtalk com seu concorrente comercial o DEc-talk. Este está baseado em regras de lingüística codificadas à mão. Não há dúvidas de que o desempenho do DEC é melhor do que o NET, mas isto deve ser visto no contexto do esforço necessário para criar cada sistema. Enquanto o NET aprendeu a partir de alguns exemplos, o DEC tem uma década de investimento em análise de lingüistas. Isto exemplifica as vantagens das RNA. Elas são fáceis de construir e podem ser usadas mesmo quando o problema não está bem entendido. Entretanto, algoritmos baseados em regras ainda são melhores do que as RNA quando suficiente entendimento está disponível.

Vejamos o comportamento do NETtalk em alguns gráficos Na figura 8 vemos como se processa o aprendizado da NETtalk. Ela primeiro aprende as saídas relativas a intensidade (as 5 últimas) e depois os fonemas (os 21 primeiros).

Na figura 9 vemos para o caso específico da letra C, como são aprendidas as sílabas. Primeiro as de som duro (/k/), e depois a de som brando /s/.

Na figura 10, temos o comportamento da rede a uma degradação intencional. Notese que diferentemente dos sistemas digitais convencionais (que desmoronam ao primeiro bit errado), a rede de agüenta sem perder muito, pelo menos no início.

Na figura 11, um aparente paradoxo. No meio do treinamento introduz-se uma pequena perturbação aleatória nos pesos da rede. Ao contrário do que diz o senso comum, a aprendizagem depois disto melhora. Uma possível explicação seria que esta chacoalhada poderia desviar mínimos locais para mínimos globais (mais profundos). Uma analogia humana seria possível. Imagine estar aprendendo línguas ouvindo uma fita gravada por uma pessoa. A perturbação consiste em trocar o locutor, e com isso seu ouvido melhora pela diversidade.

Desafio

- Localize um simulador de RN e implemente nele um reconhecedor de dígitos numéricos (de 0 a 9).

 ©88-09, Pedro Kantek

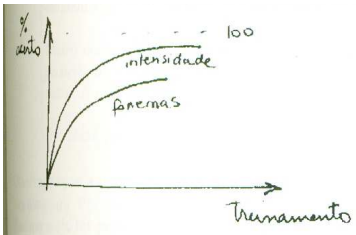


Figura 8: Ritmo de aprendizagem

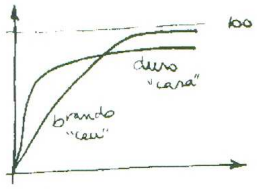


Figura 9: Diferenças entre ceu e casa

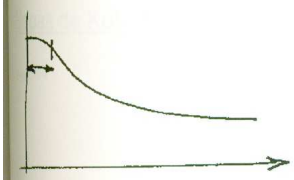


Figura 11: Degradação a alteração intencional

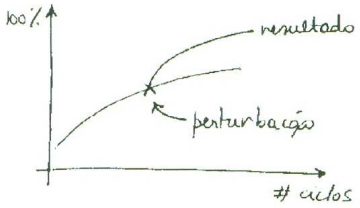


Figura 10: Evolução de aprendizagem

10

O elemento centralizador e controlador dessa verdadeira máquina animal é o cérebro, ao qual se podem atribuir as seguintes características:

O olho humano

O olho humano é a estrutura mais maravilhosamente especializada do nosso corpo. Segundo Kóvacz,

> numa proporção direta de sua importância, o sistema visual é o mais complexo, o mais estudado e o menos entendido entre os sistemas sensórios.

Sua importância transcende a própria visão. É por isso que visualizamos situações, enxergamos soluções para problemas que muitas vezes não tem sua origem no sistema sensório da visão. Também é por isso que quando um filho nos pede algo meio chato de fazer, a resposta pode ser *"vamos ver..."*.

A retina é o elemento ativo básico do olho. É onde as imagens se formam (após um fantástico processamento mecânico realizado pelo olho, como se fosse uma aperfeiçoada máquina fotográfica). Formam a retina 2 tipos de células: os cones e os bastonetes. Os cones enxergam as cores e tem resolução menor do que os bastonetes. Estes enxergam de maneira absoluta, não distinguem cores, mas tem acuidade visual várias ordens de grandeza maior que os cones.

Um bastonete responde a uma ativação de um único quanta, mostrando que olho humano atingiu o limite físico máximo de excitação. O fotopigmento responsável pela ativação dos bastonetes é a rodopsina, com resposta espectral de 400 a 600 nm. (Já se projeta construir uma memória de computador com base biológica usando a rodpsina. Veja artigo na Scientific American, vol. 272, número 3, de março de 1995: uma leitura fascinante).

As células nervosas (neurônios) da visão se dispõe em diversas camadas, como podemos ver no desenho a seguir:

Note-se como as células se dispõe em camadas. Qualquer semelhança futura com as redes neurais não será mera coincidência.

5.2 Um exemplo prático

Um exemplo de treinamento em uma rede de feedforward

Tabela 5.1: Descrição do cérebro humano

tipos de elementos de processamento	o neurônio. Conhecem-se cerca de 100 tipos de neurônio, embora não se conheça a diferença funcional entre eles
número de elementos	10.000.000.000 a 1.000.000.000.000 neurônios
tamanho / volume	volume no homem: 1500 cc, diâmetro do neurônio: cerca de 0,004 polegadas. Comprimento máximo do axônio: cerca de 1 m.
Peso	1450 gramas
Potência elétrica	10 Watts
velocidade de transmissão e chaveamento	de 10 a 100 m/seg. Chaveamento máximo: 500/Seg
Interconexão de neurônios	200.000 conexões / neurônio (nas células de Purkinje). Estas conexões são conhecidas como sinapses.
Confiabilidade	Confiabilidade do neurônio: baixa (neurônios morrem continuamente)
Confiabilidade do sistema	alta
Expectativa de funcionamento	cerca de 70 anos
Codificação da informação	digital, com modulação em freqüência

Seja uma rede em 2 camadas, contendo 3 neurônios na primeira e 2 na segunda. A rede receberá 2 entradas e gerará 2 saídas.

Suponhamos que num determinado ciclo de treinamento, a rede receba como entradas $x_1 = 0.1$ e $x_2 = 0.7$. Para estes valores de entrada, a resposta esperada é $y_1 = 0.2$ e $y_2 = 1.0$. Tudo isto é sabido de antemão.

Suponhamos ainda que os pesos da camada 1 são:

w_{ij}	b	x_1	x_2
N_1	-0.1	0.2	0.2
N_2	0.3	-0.1	0.3
N_3	0.1	0.1	0.9

E os da camada 2 são:

w_{ij}	b	N_1	N_2	N_3
N_4	0.2	0.1	-0.1	-0.1
N_5	-0.1	0.5	0.2	1.1

Segue-se agora uma série de cálculos

Cálculo da primeira camada: $U_1 = -0.1 + (0.2)(0.1) + (0.2)(0.7) = 0.06$ e $v_1 = \tanh(0.06) = 0.06$ $U_2 = 0.3 + (-0.1)(0.1) + (0.3)(0.7) = 0.5$ e $v_2 = \tanh(0.5) = 0.46$ $U_3 = 0.1 + (0.1)(0.1) + (0.9)(0.7) = 0.74$ e $v_3 = \tanh(0.74) = 0.63$

Cálculo da segunda camada:

$U_4 = 0.2 + (0.1)(0.06) + (-0.1)(0.46) + (-0.1)(0.63) = 0.097$ e $v_4 = 0.097$ (note que o neurônio 4 tem função de ativação linear, isto é $u = v$)

$U_5 = -0.1 + (0.5)(0.06) + (0.2)(0.46) + (1.1)(0.63) = 0.715$ e $v_5 = \tanh(0.715) = 0.614$

Com estes valores a saída produzida é: $y'_1 = v_4 = 0.097$
$y'_2 = v_5 = 0.614$

 ©88-09, Pedro Kantek

Conseqüentemente, os erros produzidos neste ciclo, são: $\epsilon_1 = 0.2 - 0.097 = 0.103$
$\epsilon_2 = 1 - 0.614 = 0.386$

5.2.1 O aprendizado da rede

Da diferença entre o que era esperado (0.2 e 1, neste exemplo), e a saída produzida (0.097 e 0.61, também neste exemplo), a rede começa a aprender

Conforme vemos no desenho, o aprendizado se dá quando a rede compara o resultado que ela alcançou com o que era dela esperado. Da diferença entre ambos, devidamente processada, se dá o ajuste dos valores dos pesos da rede. O algoritmo básico para este proceder atende pelo nome de *retropropagação*.

O objetivo deste algoritmo é minimizar o erro quadrático:

$$\epsilon_k^2 = (y_k - y'_k)^2$$

. Na verdade, o que se busca é a minimização do erro médio quadrático

$$F = E(\epsilon^2) = \frac{1}{p}.\sum_{k=1}^{p} \epsilon_k^2$$

. O método para este proceder é o do gradiente descendente [1] e pode ser assim discriminado:

- a nova direção do aprendizado é dado pela direção antiga mais um δ
- o δ é obtido pelo produto de um fator de aprendizagem α multiplicado pelo gradiente no ponto w
- o gradiente no ponto w, é a derivada parcial do erro médio quadrático

Vejamos a seguir, o desenvolvimento matemático do que acima se disse:

$$W = W + \Delta W$$

$$\Delta W = -\alpha \nabla_w F$$

$$\Delta_w F = \frac{\partial F}{\partial W}$$

$$F = (\epsilon^2)$$

$$\Delta w_{ij} = -2\alpha v_j \delta_i$$

onde w_{ij} = variação do peso ij
α = coeficiente de aprendizado ($0.1 \leq \alpha \leq 0.05$)
v_j = valor original da saída do neurônio j
δ_i = erro retropropagado.

Com essa base, podemos descrever o algoritmo de retropropagação

[1] o gradiente de uma função em um ponto tem a seguinte fórmula

$$\nabla U = \frac{\partial U}{\partial x} i + \frac{\partial U}{\partial y} j + \frac{\partial U}{\partial z} k$$

e tem um conceito bem simples: imagine-se uma superfície irregular. Suponha-se um ponto qualquer sobre essa superfície. A partir desse ponto deve-se desenhar inúmeras semi retas, com origem no ponto e em todas as direções possíveis. A semi-reta que mais apontar para cima, é a direção do gradiente daquela função (representada pela superfície) naquele ponto.

5.2.2 Algoritmo de retropropagação (backpropagation)

1. Obter as derivadas de $f(v)$. Naturalmente, estas derivadas dependem de qual é a função de ativação que se está empregando. A derivada da função linear $u = v$ é 1. A derivada de $\tanh(u)$ é $1 - u^2$.

2. Inverter o sentido do treinamento (das saídas para as entradas, daí o "retro"do nome)

3. Retropropagar o erro, de acordo com o desenvolvimento acima listado

4. Calcular os novos pesos.

No exemplo que estamos a estudar, fica
Para $v = u$, $\frac{dv}{du} = 1$, para $v = \tanh(u)$, $\frac{dv}{du} = 1 - u^2$.

As derivadas, para cada um dos neurônios da nossa rede exemplo, são:

Neurônio N_i	Derivada de v
N_1	$1 - (0.06)^2 = 1$
N_2	$1 - (0.46)^2 = 0.79$
N_3	$1 - (0.63)^2 = 0.6$
N_4	1
N_5	$1 - (0.61)^2 = 0.63$

Os resultados numéricos são:
$\epsilon_4 = (0.103)(1) = 0.103$
$\epsilon_5 = (0.386)(0.63) = 0.24$
$\epsilon_1 = [(0.103)(0.1) + (0.24)(0.5)] \times 1 = 0.13$
$\epsilon_2 = [(0.103)(-0.1) + (0.24)(0.2)] \times 0.79 = 0.03$
$\epsilon_3 = [(0.103)(-0.1) + (0.24)(1.1)] \times 0.6 = 0.152$

Estes novos valores vão substituir os pesos anteriores. Este ciclo ocorrendo inúmeras vezes, vai diminuindo cada vez mais a diferença entre o real e o esperado. A partir de um certo ponto, a rede... "aprendeu".

Em resumo

No exemplo acima, a saída real do N_4 é $y_4 = 0.097$, quando deveria ser $y'_4 = 0.2$. O neurônio N_5 deu $y_5 = 0.613$ quando deveria dar $y'_5 = 1.0$.

Com isto temos os erros $\epsilon_4 = 0.103$ e $\epsilon_5 = 0.386$.

O que se quer minimizar é o erro médio quadrático (médio = atingir todos os neurônios, quadrático = a fim de que erros opostos não se anulem).

O gradiente em um ponto é a direção que maximiza (ou minimiza) o erro, e em termos analíticos é a derivada parcial do erro médio quadrático.

A nova direção é o gradiente no ponto × fator de aprendizagem

A correção do peso é igual ao peso antigo × a nova direção

5.2.3 Um exemplo real

A Companhia Siderúrgica Nacional (CSN), tem um alto-forno para produzir aço. Nesse forno, sempre entram os seguintes insumos:

- Ferro gusa (medido em toneladas), a uma determinada temperatura (medida em graus centígrados)

- Sucata de minério (medido em toneladas)

 ©88-09, Pedro Kantek

- Minério bruto (medido em toneladas)
- Calcita (medido em toneladas)
- Oxigênio injetado no forno, por um certo intervalo de tempo (medido em minutos)

Desse processamento, saí um determinado aço, com certa percentagem de Carbono (medido em %) e de enxofre (também medido em %).

Existe em operação um sistema fenomenológico (isto é, baseado em fórmulas físicas), que orienta o processo. Como se verá adiante, este sistema é falho e muitas vezes o operador acaba se baseando muito mais em sua experiência (aprendizado) do que no sistema. É graças a esta característica (existe um conhecimento intrínseco ao problema, que não é mostrado pelas fórmulas físicas e que reside nos neurônios do operador) que se pode buscar uma rede neural artificial para substituir o processo fenomenológico.

Em geral, entram no forno cerca de 200T de ferro gusa a 1.700 graus centígrados, bem como partes menores dos demais componentes. Injeta-se oxigênio (o chamado "sopro") durante cerca de 15 minutos. Ao final deste tempo, retira-se uma amostra e é feita uma análise ultra-rápida, antes que a tralha toda resfrie. Dependendo do resultado, pode ou não haver a necessidade de um ressopro, ou seja, a injeção de mais oxigênio.

Em média, pelo processo atual, em cerca de 20% dos casos é necessário o ressopro, e isto custa caro, pois demora mais tempo para a produção do aço, e principalmente desgasta o forno conversor. Há ainda um risco pior ainda, quando a coisa desanda: a chamada "projeção"que nada mais é do que um transbordo do material ígneo. Isso ocorre quando formam-se bolhas no forno e quando elas sobem e arrebentam, é um salve-se quem puder. Algo parecido acontece quando se frita um ovo que acabou de sair da geladeira e se está sem camisa. Imagine o que acontece no forno de aço, quando em vez de micro-gotículas de óleo quente são macro-gotas de ferro gusa a 1700 graus.

Criou-se uma RNA com o objetivo de reduzir a zero a necessidade do ressopro e eliminar o risco da projeção. Vejamos como a coisa foi feita:

- Levantaram-se na CSN, os dados de cerca de 3.000 rodadas passadas (dados históricos) referentes a ciclos de produção de aço. Os dados se compunham de:
  - Entradas: P(gusa), C(gusa), P(sucata), P(minério), P(calcita) e T(oxigênio)
  - Saídas: C(gusa), %(carbono), %(enxofre) e Projeção(sim/não)
- Desse levantamento desprezaram-se cerca de 700 amostras, por incompletude ou inconsistência dos dados. Sobraram, em conseqüência, 2.300 amostras.
- Estas, foram divididas em 2 universos: O universo de treinamento (1.600) e dados de teste (700).

A seguir, os seguintes passos foram dados:

Normalização das variáveis

As variáveis x_i (entrada) e y_i (saída) foram normalizadas. Se x_i e y_i são contínuos, faz-se uma transformação algébrica para que variem no intervalo $-1 \leq x_i, y_i \leq 1$.

Se x_i e y_i são lógicos, a transformação é tal que $x_i, y_i \in \{-1, 1\}$

No exemplo, a P(gusa) era de 200T, com variação para mais ou menos de 10%. A normalização fez x_i variar entre -1 e 1, com limites reais de 180 e 220.

Compactação do problema

O objetivo desta fase é reduzir a dimensão do problema. Informações redundantes devem ser pesquisadas, e se encontradas, eliminadas. Busca-se como objetivo ter um número máximo de variáveis igual a 100.

Interpretação das saídas lógicas

Se $y_i \in \{-1, 1\}$ e $y'_i = \tanh u_i$, isto implica que $-1 \leq y'_i \leq 1$, e então se $y'_i > 0$, a resposta considerada será 1 e se $y'_i < 0$, a saída considerada será -1. Aqui, um detalhe importante: esta convenção de respostas lógicas é feita apenas em tempo de operação e não em tempo de treinamento, sob pena de enlouquecer a rede.

Dimensionamento da rede

Tipo de neurônio Na camada de saída, considera-se a regra: Se y_i é contínuo, usa-se a função linear ($v = u$). Se, ao contrário, y_i é lógico, usa-se a função tangente hiperbólica ($v = \tanh u$). Nas camadas intermediárias, usa-se sempre a função tangente hiperbólica.

Número de camadas Considerar-se-ão os seguintes dados: $x \in R^n, y \in R^m$, e $n > m$. Começa-se fazendo uma tentativa com uma única camada de n entradas e m neurônios. Segue-se um treinamento, teste, e análise do desempenho. Se satisfatório, OK o problema está resolvido. Se não, parte-se para a solução de 2 camadas.

Esta contém n entradas, m saídas e p neurônios ocultos. O valor inicial de p é a média entre m e n. Segue-se o mesmo processo: treinamento, teste, e análise do desempenho. Se satisfatório, vai-se diminuindo p, até encontrar o valor mínimo para ele que gere desempenho da RNA aceitável. Se, de cara, o desempenho não é satisfatório, aumenta-se progressivamente o valor p.

Inicialização da rede

Escolhem-se w_{ij} e b_i de maneira randômica, em geral com seus valores variando entre -0.1 e 0.1.

Escolha do fator de aprendizagem

Se α é grande, a rede pode divergir, e o faz com freqüência. Se α é pequeno, o processo torna-se muito lento. Em geral escolhe-se o α variando entre 0.001 e 0.2, sendo um $\alpha_{tipico} = 0.05$ ou 0.1. Uma possibilidade bem interessante é a utilização de um fator variável, sendo maior no começo do processo, para acelerar a convergência e menor ao final, para garantir o ajuste fino. Isso lembra um pouco a computação evolutiva, só que lá o operador crossover faz o papel de α grande e o operador mutação o de α pequeno.

O problema de manter o α grande em todo o processo, faz com que neste ponto, por exemplo, o ponto objetivo fique pulando de um lado para o outro, sem se aproximar do mínimo (ou máximo) absoluto.

Treinamento e teste

O conjunto de pares entrada-saída (x^k, y^k) deve ser estatisticamente representativo de seu universo. Lembremos que a rede aprende o que lhe é ensinado. A divisão entre treinamento e teste pode ser assim feita:

$$(x^k, y^k)_{k=1,2...,p} \{ \frac{(x^k, y^k)_{k=1,2...,p/2} => treinamento}{(x^k, y^k)_{k=p/2,..,p} => teste} \}$$

5.2.4 Fenômeno do super-aprendizado

O fenômeno é bem simples: durante o treinamento chega um ponto em que há que interrompê-lo. Se isso não for feito, a rede se vicia nos dados de treinamento e deixa

 ©88-09, Pedro Kantek

de responder satisfatoriamente aos dados do ambiente - ainda desconhecido - com que defrontará depois.

A rede fica viciada nos dados de treinamento, e à eles responde com perfeição. Entretanto seu desempenho no geral começa a decair. Essa é a finalidade de se dividir a massa de amostras em 2 partes: a rede é treinada com uma delas e depois testada com a outra. Quem manda a hora de parar o treinamento é a segunda amostra, e não a primeira.

Se o treinamento não for interrompido, o desempenho da rede segue a linha inferior. Cada vez o erro é menor, aproximando-se no caso de treinamento ad eternum assintóticamente do zero erro.

O problema é que este zero erro refere-se apenas aos dados de treinamento. Desde o começo, o desempenho da rede sobre os dois conjuntos anda mais ou menos parelho: enquanto a rede é treinada ela atende bem ao conjunto de teste.

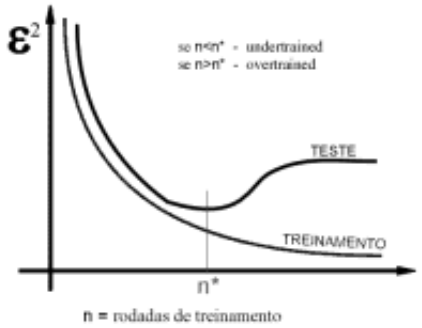


Figura 5.3: Fenômeno do super-treinamento em redes neurais

A partir de um certo ponto, identificado com n* na figura 5.3, se o treinamento prossegue, o desempenho nos dados de teste (que são os que interessam: eles representam o universo), começa a piorar. Note como a curva do erro sobe a partir deste ponto.

Desafio

- Localize um simulador de RN e implemente nele um reconhecedor de dígitos numéricos (de 0 a 9).

Capítulo 6

Computação Evolutiva

6.1 Introdução

6.1.1 Resolvendo problemas

Podemos definir o ser humano como um resolvedor de problemas.[1] De fato, sua sobrevivência e o progresso da humanidade tem sido uma luta constante contra dificuldades de toda ordem. O que distingue o homem dos demais animais é este aspecto: a racionalidade, que lhe permite pensar mais a frente.

Nesta figura a classe de problemas 1 (a mais externa) engloba questões que sequer chegamos a considerar como problemas, tal a dimensão por elas assumida e a dificuldade de vislumbrar uma proposta de abordagem. Para esta classe, não se conhecem métodos de tratamento nem critérios para conhecer o sucesso ou insucesso de uma possível proposta de ataque. Podemos citar como exemplo, desta classe, a quantificação de quantos planetas tem vida inteligente no universo, a individualização de uma célula reprodutiva masculina humana no ato da concepção (quem será o vencedor da corrida ?), o que acontecerá com o planeta terra daqui há 300 séculos,...

A classe seguinte, (incluída na classe 1), apresenta problemas, que se ainda são formidáveis, pelo menos permitem algum tipo de tratamento. São em geral problemas com pouca ou nenhuma retroalimentação (o que permitirá nas outras classes quantificar o sucesso na tentativa de solucionar o problema). São exemplos desta classe, a educação de filhos. Outro bom exemplar é o célebre "último teorema de Fermat ".

A terceira classe, já permite um tratamento mais formal às soluções. A questão nesta classe não é qualitativa, mas quantitativa. Teoricamente, se tivessemos tempo infinito, poderíamos solucionar todos os problemas desta classe. Exemplo aqui é o jogo de xadrez . Se limitarmos o comprimento dos jogos a 40 lances (ou 400, se se achar 40 pouco), estipulando-se o empate, caso nenhum dos contedores tenha vencido neste limite, podemos afirmar sem medo de errar, ser o jogo um problema finito. E mais, facilmente resolvível, posto que dada uma situação no tabuleiro, qualquer bom jogador é capaz de responder, senão o lance genial e inesperado, pelo menos o lance que menos perde, ou o mais razoável de ser ofertado naquele momento. A questão é que imagina-se existirem 10^{120} jogos distintos, e se fossemos analisar todos eles, só conseguiríamos tendo tempo infinito a nossa disposição.

Finalmente, a quarta classe é a dos problemas computáveis. Desde os simples, como resolver uma equação do segundo grau, até os verdadeiramente complexos como otimizar uma função linear de 10.000 variáveis e 20.000 restrições, por exemplo.

Há que se ter em mente algumas questões, como

[1] e o programador como um meta-resolvedor de problemas, mas isso é outra história

- A separação entre as classes não é caso resolvido. Certos problemas podem ora estar numa classe ora na outra, dependendo do volume de dados e dos recursos à disposição para sua solução.

- Também os limites variam no tempo. À medida em que a ciência e a técnica progridem, existe uma tendência dos problemas de ocuparem classes crescentes (da 1 para a 2, desta para a 3 e assim por diante).

6.2 Genética

Antes de estudar os algoritmos genéticos, precisamos estudar e conhecer o modelo original. O problema de descobrir como os seres vivos se criam na natureza sempre ocupou um lugar importante nas especulações do intelecto humano.

6.2.1 Gregor Mendel

Tudo começa quando Mendel publica em 1866 (na verdade os trabalhos foram primeiramente lidos na Sociedade Histórica Natural de Brünn em 8 de fevereiro e 8 de março de 1865). O texto tinha as conclusões de suas observações junto a cruzamentos efetuados com ervilhas. Segundo Mendel, a herança se dá através de caracteres particulados (um recebido de cada pai) e apenas um destes caracteres sobrevém no filho, embora o outro permaneça lá pronto para ser usado nas próximas descendências.

Com esta explicação Mendel conseguiu mostrar como os filhos heradavm características dos pais, sem no entanto levar à pasteurização (homogeneização) da descendência. Isto é o que diz a Primeira lei de Mendel, ou Princípio da segregação:

Padrões hereditários são determinados por gens que ocorrem em pares no indivíduo, mas que segregam um ao outro na formação dos gametas, de modo que qualquer gameta recebe um o outro dos alelos pareados. O número duplo de genes é reestabelecido na prole.

Experiência similar à de Mendel Suponhamos a colus blumei , planta caseira que pode ter 2 tipos de folha: crenada e recortada. A borda da folha crenada é regular quase reta, e a da recortada, traz profundos recortes. Esta planta se reproduz por cruzamento ou por autopolinização. Comecemos separando uma planta crenada e uma recortada. Façamos séries de autopolinização , e garantamos que os filhos sempre serão iguais aos pais. Com isto podemos garantir que tais plantas são de raça pura para este caracter (um geneticista usaria o termo homozigoto).

Agora cruzamos as duas: todos os filhos de ambas tem folhas recortadas (dominante). Cruzando os filhos destes, temos 1/4 de crenadas (recessivo) e 3/4 de recortadas. Note-se que este resultado se mantém, mesmo que o recortado seja o genitor pistilado (mãe, fornece o óvulo), ou o genitor estaminado (pai, fornece o gameta masculino).

O trabalho de Mendel só foi redescoberto neste século em trabalhos independentes de Vries , Correns e Von Tschermak . Em 1902, Sutton propôs que os fatores de Mendel estivessem fisicamente localizados nos cromossomos. Ele afirmou isso se baseando no comportamento dos cromossomos durante a divisão celular (mitose e meiose). Nos anos 40, Avery estabeleceu que os genes (cromossomos) eram compostos de um ácido nucleico (DNA = desoxirribonucleic acid). Também nos anos 40, Tatum demonstrou que a falta de cor em olhos de Drosphila era devido a um defeito na etapa de biossintese do pigmento. Cada etapa na síntese do pigmento era devida a um gene específico. Isto levou à lei "Um gene-uma enzima", que propõe que cada gene gera uma enzima (proteina) que cataliza uma determinada reação química. Ou seja, é importante lembrar que o que é

transmitido de pai para filho não é um caráter em si, mas sim algo que determine o posterior desenvolvimento deste caracter em lugar, hora e ambiente apropriado.

Em 53, Watson e Crick, em artigo na Nature propuseram a estrutura de duplo hélice do DNA (e ganharam um Nobel por isso).

6.2.2 DNA

Substância orgânica, encontrada nas células vivas, no qual as informações hereditárias estão armazenadas e a partir de onde podem ser transferidas. O DNA forma longas cadeias, compostas de nucleotídeos, cada um contendo uma de quatro bases: adenina (A), guanina (G), citosina (C) e tiamina (T)

Cada seqüência de 3 nucleotídeos especifica um tipo particular de amino-ácido. Existem cerca de 20 amino-ácidos conhecidos. A adenina só se liga à Tiamina, e a citosina só se liga à guanina. Temos então 4 possibilidades: AT, TA, CG, GC. Como cada grupo de 3 ligações gera um amino-ácido, temos um total de $4^3 = 64$ possibilidades. Na verdade, muitas delas são redundantes (2 seqüências diferentes geram o mesmo amino ácido).

Na tabela a seguir, aparece a lista dos 20 aminoácidos bem como sua regra de constituição, na **combinação do RNA**. A base na vertical esquerda é a primeira, no alto da tabela é a segunda e na vertical direita, a terceira. Os elaboradores deste dicionário, Marshall Nirenberg e Gobing Khorana (além de Holley, por seu trabalho com tRNA) receberam o Prêmio Nobel de Medicina e Fisiologia em 1968.

	U	C	A	G	
U	Fenilalanina	Serina	Tirosina	Cisteína	U
U	Fenilalanina	Serina	Tirosina	Cisteína	C
U	Leucina	Serina	STOP	STOP	A
U	Leucina	Serina	STOP	Triptófano	G
C	Leucina	Prolina	Histidina	Arginina	U
C	Leucina	Prolina	Histidina	Arginina	C
C	Leucina	Prolina	Glutamina	Arginina	A
C	Leucina	Prolina	Glutamina	Arginina	G
A	Isoleucina	Treonina	Asparagina	Serina	U
A	Isoleucina	Treonina	Asparagina	Serina	C
A	Isoleucina	Treonina	Lisina	Argenina	A
A	Metionina ou START	Treonina	Asparagina	Serina	G
G	Valina	Alanina	Ác. aspártico	Glicina	U
G	Valina	Alanina	Ác. aspártico	Glicina	C
G	Valina	Alanina	Ác. glutâmico	Glicina	A
G	Valina ou START	Alanina	Ác. glutâmico	Glicina	G

Duplicação do DNA

A cópia exata da molécula de DNA é provida antes da divisão celular. As espirais da hélice se separam, e cada ramo antigo cria um complementar completo, ligando-se a bases complementares. Ao final temos 2 hélices duplas.

Síntese de proteinas A cadeia de nucleotídeos do DNA é transcrita para uma seqüência de nucleotídeos do RNA-m (mensageiro) (O RNA contém o açucar ribose no lugar da dexorribose, e a base uracil no lugar da tiamina). O RNA-m é o complementar do DNA, e é formado por trios de nucleotídeos, chamados codons. A partir do RNA-m entra em ação o RNA-t (de transferência) é formado por anti-codons do RNA-m (e portanto iguais ao DNA). Acompanhe no exemplo:

 ©88-09, Pedro Kantek

Molécula de DNA	CAA	CTC	TTT
Molécula de RNA-m	GUU	GAG	AAA
Molécula de RNA-t	CAA	CUC	UUU
Proteína	Valina	Ácido Glutâmico	Lisina

Meiose É o processo de divisão celular no qual os cromossomos são divididos à metade do seu número original. A meiose ocorre apenas durante a formação das células sexuais (óvulos e espermatozóides). Uma célula qualquer do corpo (somática) contém um par de cada tipo de cromossomo, ou seja é diploide. A meiose produz células com apenas 1 elemento de cada par (haplóide). Na fertilização, duas células haplóides são unidas e com isso, seu resultado (o ovo, ou zigoto) volta a ter um número diplóide.

Mitose Neste tipo de divisão, que ocorre em todas as células vivas não sexuais, os cromossomos são replicados exatamente e as duas partes (iguais) são distribuidas igualmente às duas células filhas.

6.2.3 Evolução

Conceito embutida na crença de que os organismos existentes descendem de um ancestral comum. Esta teoria, também conhecida como descendência com modificação constitui a evolução orgânica. A evolução inorgânica trata do universo físico, a partir da matéria original. A evolução orgânica concebe a vida como tendo começado em um organismo simples, composto de massa protoplásmica primordial, do qual se originaram ao longo do tempo todas as formas de vida. A primeira teoria da evolução foi proposta por Jean Lamarck em 1801 e incluia a herança de características adquiridas como sendo a força operativa na evolução. Posteriormente, em 1858, Darwin e Wallace descreveram sua teoria da evolução baseada na seleção natural, focada na sobrevivcência e reprodução das espécies melhor adaptadas ao ambiente. Esta teoria teve um efeito profundo sobre o pensamento científico. Embora tenha sido aprimorada por desenvolvimentos científicos posteriores, a teoria da evolução permanece essencialmente a mesma que foi sugerida por Darwin, e comprovada pela genética, bem como anatomia comparada, embriologia, geografia, paleontologia e mais recentemente bioquímica. [MIC92]

Haemophilus influenzae Rd Esta bactéria teve seu seqüenciamento completo terminado por Fleischmann e seus colaboradores em 1995. Seu genoma consta de 1.830.137 pares de bases. Isso equivale a aproximadamente 3.6×10^6 bits, quase meio megabyte. Ainda não se conhece a função de cada um de seus 1.743 genes a aplicação de técnicas conhecidas pelos engenheiros (diagramas de circuitos lógicos) estão sendo úteis para entender o funcionamento do organismo.

Charles Darwin Naturalista inglês, (1809-1882). Durante 6 anos, entre 1831 e 1836 atuou como naturalista oficial a bordo do HMS Beagle, durante as pesquisas que este fêz nas costas da América. A viagem toda acabou sendo uma volta ao mundo. Durante a viagem, Darwin começou a acumular dados e a pensar no conceito de evolução. De volta à Inglaterra, ele tinha dúvidas se publicava ou não suas conclusões, até que recebendo uma correspondência de Alfred Wallace, que lhe expunha idéias muito parecidas com as suas, acabou decidido a publicar seus pensamentos. Hoje a história registra a ambos como pais da idéia da seleção natural.

6.3 Algoritmos genéticos

Algoritmos genéticos foram inventados na tentativa de imitar os processos envolvidos na seleção natural por John Holland em 1975. Desde que a teoria da evolução foi aceita, os biólogos estão maravilhados com a vida e seu nível de complexidade. Os mecanismos que dirigem tal evolução não estão completamente conhecidos, mas algumas de suas características sim. A evolução acontece nos cromossomos – unidades orgânicas que codificam a estrutura da vida. O processo específico de codificação e decodificação dos cromossomos não é conhecido, porém alguns detalhes da teoria tem larga aceitação: A evolução opera nos cromossomos e não na vida que eles codificam A seleção natural é a ligação entre cromossomos e o desempenho das estruturas que eles decodificam. Os processos de seleção natural fazem com que cromossomas que codificam estruturas bem sucedidas se reproduzam mais freqüentemente do que as que não tem sucesso.

O processo da reprodução é o ponto no qual a evolução tem lugar. A mutação pode causar que os cromossomas dos filhos sejam diferentes daqueles de seus pais (biológicos). A recombinação pode criar cromossomos completamente diferentes pela combinação do material dos cromossomos dos seus pais.

A evolução biológica não tem memória. Tudo quanto é sabido para produzir indivíduos que funcionem bem em seu habitat está contido na bagagem de gens – o conjunto de cromossomos carregados por um indivíduo.

Holland acreditava que apropriadamente incorporados a algoritmos ele poderia obter uma técnica para resolver resolver problemas difíceis como a natureza tinha feito: através da evolução. Ele começou a trabalhar com algoritmos que manipulavam strings de dígitos binários a quem ele chamou de cromossomos. Como a natureza, estes algoritmos resolviam o problema de encontrar bons cromossomos pela manipulação dos cromossomos cegamente.

Como a natureza, eles nada sabiam sobre o problema a resolver. A única informação que eles obtém é uma avaliação de cada cromossomo que é produzido e usa-se apenas esta avaliação para valorar a seleção de cromossomos de tal maneira que aqueles com boa avaliação tendem a se reproduzir mais do que aqueles com avaliação ruim.

Estes algoritmos usando mecanismos de codificação e reprodução simples exibem um comportamento complicado e resolvem problemas extremamente complexos.

Deve-se ter em conta que embora geneticistas e biólogos tenham influenciado o campo de algoritmos genéticos e continuem a fazê-lo, esta influência é unidirecional. (Aqui o Davis diz que junto com redes neurais e *simulated annealing* estas disciplinas só obtém uma inspiração inicial da natureza e depois seguem separadas. Não é a opinião de Winston que sugere que IA deve ser estudada por cientistas de computação (tornar os computadores mais usáveis) e por biólogos, psicólogos e lingüistas (conhecer melhor o funcionamento da natureza).

6.4 Vocação

Começa-se citando Goldberg que em 1989, escreveu *"algoritmos evolutivos podem ser aplicados a virtualmente qualquer problema"*. Entretanto, olhando sob um prisma um pouco mais restrito, pode-se afirmar que a Computação Evolutiva (CE) tem a habilidade de manusear problemas complexos. Mesmo sem oferecer respostas absolutamente corretas (até pela complexidade dos problemas), ela sempre tem algo a oferecer na busca de resultados para este tipo de questão. Mais especificamente ainda falando, como saber se para um dado problema a computação evolutiva tem algo a oferecer ? Não há resposta definitiva, mas parece haver consenso na literatura sobre o tema de que problemas com algumas das seguintes características:

 ©88-09, Pedro Kantek

- Espaço de busca muito grande (o que inviabiliza a busca exaustiva) ou espaço de busca não completamente conhecido
- Função de avaliação sujeita a ruídos ambientais, ou não completamente conhecida
- Não necessidade da obtenção de um máximo global, sendo satisfatório um máximo local ainda que sujeito a algum critério de aceitação
- Espaço de busca multimodal

poderiam ser satisfatoriamente abordados usando ferramentas da CE. Note-se que este enfoque pressupõe CE "pura", sem hibridização. A partir do instante que se agrega a esta abordagem um conhecimento advindo do problema, pode-se generalizar a vocação da CE, e fazer coro a Goldberg quando este diz que qualquer problema pode ser tratado usando CE. Restaria apenas a questão de "quanto"do conhecimento do problema deve ser agregado.

6.4.1 Conceituação

A computação evolutiva é um ramo da ciência da computação que propõe um paradigma alternativo ao processamento de dados convencional. Enquanto neste, antes de se lançar mão de um computador para resolver ou auxiliar na solução de uma determinada questão, há que se conhecer previamente a maneira de encontrar a solução, na computação evolutiva este conhecimento pode ser prescindido.

Na CE (computação evolutiva), tal conhecimento é substituído por uma simulação do mecanismo da evolução natural. Como já descreveu Darwin, a vida em nosso planeta é o resultado de um processo aparentemente aleatório, no qual o meio ambiente exerceu e exerce um papel de modelador e de árbitro do resultado. A constatação da diversidade da vida, associada ao fato de que todos os seres vivos compartilham uma bagagem genética comum, pelo menos em termos de seus componentes básicos, fornece bem uma idéia de quão rica e multifacetada tem sido a atuação do meio ambiente na condução do processo de manutenção e melhoria da vida.

A CE está baseada em quatro processos fundamentais: reprodução, variação randômica, competição e seleção de indivíduos dentro da população. Segundo Atmar, citado em Fogel, *"eles formam a essência da evolução, e sempre que estes quatro processos surgem, seja dentro de um computador, seja na natureza, a evolução inevitavelmente aparece"*.

A localização da CE no espectro da ciência da computação ainda é controvertida, provavelmente pelo pouco tempo decorrido desde a publicação de suas idéias fundamentais. Mais ainda, tal ciência não teve origem única, surgindo variantes ao longo dos anos 60-80, em situações diferentes, em países diferentes e para tentar resolver problemas diferentes. Só agora, no início do século XXI, é que se assiste uma tentativa de unificação de tais variantes, buscando-se uma dogmatização que consolide e selecione as principais idéias do campo.

Possivelmente, a melhor inserção da CE na ciência da computação seja aquela descrita em Heitkoetter e Beasley, a qual deve ser vista, não como posição definitiva (que de resto não existe ainda), mas apenas como uma proposta de classificação. Segundo Heitkoetter, o maior ramo de classificação que abrange a CE é o que foi chamado de "Computação Natural". Ele inclui os tópicos de vida artificial, geometria fractal, sistemas complexos e um ramo que foi identificado como inteligência computacional. Este último inclui redes neurais artificiais, sistemas nebulosos e a computação evolutiva. A seguir, tem-se o aspecto visual desta proposta de classificação

- ...Outros ramos da ciência da computação

- Computação natural
  - Vida artificial
  - Geometria fractal
  - Sistemas complexos
  - Inteligência computacional
    1. Redes neurais artificiais
    2. Sistemas nebulosos
    3. Computação evolutiva

O próprio termo "computação evolutiva"é novo, ele começa a ser usado apenas a partir de 1991. A CE está baseada em algumas idéias básicas, que quando implementadas, permitem simular o processo de passagem de gerações da evolução natural dentro de um computador. Pode-se discutir se a seleção natural tem ou não um objetivo, se o universo e os seres vivos fazem parte de um plano ou são mero acaso, discussão que não cabe aqui, mas se a seleção natural tivesse um "objetivo", este poderia ser resumido como a expansão e melhoria da vida. Na CE tal objetivo deve ser substituído por aquele que for mais adequado ao problema que se está a estudar. Feita essa mudança de enfoque, as idéias que permitem a simulação de que se fala acima são:

- A criação de uma população de soluções, possivelmente obtida na sua primeira geração de modo aleatório, e na qual os indivíduos tenham registrado de modo intrínseco os parâmetros para obter uma solução ao problema posto.

- A criação de uma entidade que possa fazer julgamentos quanto à aptidão de cada um dos indivíduos. Essa entidade pode assumir as mais diversas formas (um conjunto de funções matemáticas, um programa de computador, um avaliador de desempenho, entre outros), e ela não precisa deter conhecimento de como encontrar uma solução para o problema, mas apenas de atribuir uma "nota" ao desempenho de cada um dos indivíduos da população.

- E finalmente, mas não menos importante, a criação de uma série de operadores que serão aplicados à população de uma dada geração para obter os indivíduos da próxima geração. Tal série de operadores, é copiada (pelo menos em sua concepção filosófica) da natureza. Os principais operadores citados na literatura são:

  seleção permite escolher um indivíduo ou um par deles para gerar descendência. Note-se que este operador simula a reprodução assexuada (no primeiro caso) e a sexuada (no segundo) que ocorrem na natureza. Obviamente, a prioridade da escolha recai sobre indivíduos mais bem avaliados pela entidade de avaliação. Note-se que a seleção não cria novas soluções, apenas indica, das existentes, qual (is) a (s) melhor (es).

  recombinação operador que simula a troca de material genético entre os ancestrais para a geração da prole. Basicamente o conceito é fácil de entender: se há dois indivíduos que são bem avaliados, embora por duas razões distintas, a recombinação poderia, em tese, criar um descendente que juntasse essas duas razões em um único indivíduo que seria melhor que seus ancestrais.

  mutação operador que introduz mudanças aleatórias sobre o material genético.

A CE se divide em seus diversos (sub)campos de acordo com a importância e a operacionalização dos conceitos vistos acima, quando de sua implementação em modelos computacionais. A figura 2, mostra uma possível distribuição de tais subcampos.

 ©88-09, Pedro Kantek

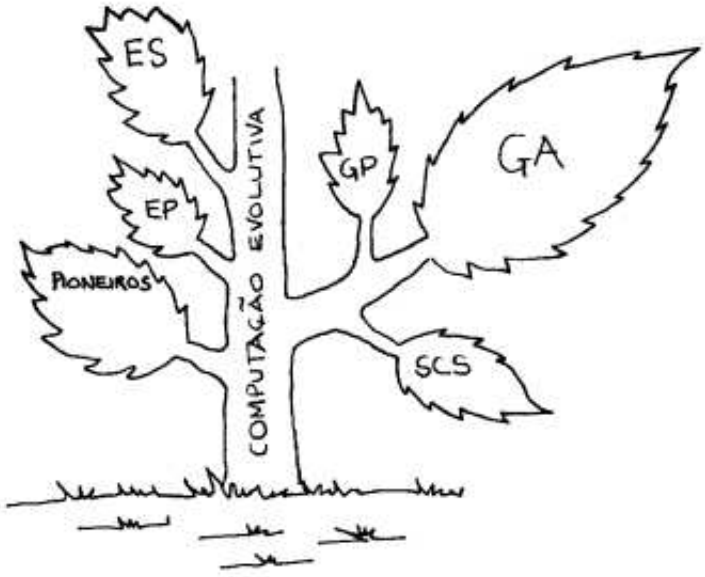


Figura 6.1: A Computação Evolutiva e seus campos

Cada uma das folhas da árvore na figura 6.1 representa uma tendência ou uma concepção prévia a respeito de algumas características que a simulação da evolução natural deve ter. Historicamente, as primeiras iniciativas na área (representadas pela folha chamada de "pioneiros"), são de biólogos e geneticistas interessados em simular os processos vitais em computador, o que recebeu na época o nome de "processos genéticos". Alguns desses cientistas são: Barricelli, 1957, 1962; Fraser, 1960, 1962; Martin e Cockerham, 1960. Um biólogo, Rosenberg, em 1967 ao escrever sua tese de doutorado, simulou uma população de seres unicelulares, estrutura genética clássica (um gene, uma enzima), com estrutura diplóide, com cromossomos de 20 genes e 16 alelos permitidos em cada um. Já na década de 60, Holland e outros começaram a estudar os chamados sistemas adaptativos, que foram modelados como sistemas de aprendizagem de máquina. Tais modelos conhecidos como algoritmos genéticos (GA na figura 6.1 implementavam populações de indivíduos contendo um genótipo, formado por cromossomos (que neste modelo eram representados por seqüências de bits), e aos quais se aplicavam os operadores acima citados: seleção, recombinação e mutação. Ainda que Holland tenha proposto um quarto operador (a inversão), este não chegou a ser largamente usado. Uma das primeiras aplicações propostas para os algoritmos genéticos (cuja primazia no uso do termo cabe a Bagley na sua dissertação em 1967), seguindo o caminho pesquisado por Holland, foram os sistemas classificadores (identificados na figura 6.1 pela folha contendo SCS).

Sistemas classificadores O pesquisador John Holland, logo no início do seu trabalho com Algoritmos Genéticos, propôs este tipo de sistema, ao qual chamou de "Sistemas Classificadores". A ênfase aqui, era a habilidade de aprender. Logo, a palavra aprender (learn) foi incluída no nome do sistema e eles ficaram como "Learning Classifier System", ou sua abreviatura, LCSs. Em outubro de 1992, ocorreu o primeiro workshop em LCS no Centro Espacial Johnson da NASA em Houston, Texas.

Holland começa definindo uma "caixa preta". Esta, pode ser pensada como um computador. Ele tem as suas entradas e saídas, além de um sistema de mensagens passando entre estes dois limites. Este sistema converte (computa) mensagens de entrada em mensagens de saída, de acordo com um conjunto de regras, também conhecidas como

"programa de computador".

Da teoria da computação, vejamos a mais simples de todas as estruturas de regras, chamadas sistemas de produção. (SP). Os SPs já foram demonstrados como computacionalmente completos por Post (em 1943), razão pela qual são às vezes conhecidos como Sistemas de Post.

Chamando a essa regra "se...então..." de classificador e escolhendo uma representação que a torne fácil de operar (por exemplo trabalhando com strings binários) podemos ter uma população de classificadores e imediatamente saberemos como gerar novas regras para o nosso sistema: usando um algoritmo genético para gerar novas regras/classificadores a partir da população corrente.

Tudo são mensagens compostas de 0´s e 1´s fluindo dentro da caixa preta. Elas são mantidas em uma estrutura chamada lista de mensagens. Olhemos agora dentro da caixa preta. A interface com a entrada gera mensagens (strings binários) que são escritos na lista de mensagens. Agora, as mensagens da lista são comparadas com a parte da condição de todos os classificadores, para descobrir quais ações poderiam ser disparadas. A lista de mensagens é esvaziada e as ações (codificadas como mensagens) são colocadas na lista de mensagens. Agora a interface com a saída verifica as mensagens para atuar sobre os efetores. E, o ciclo recomeça.

Note que existe a possibilidade de existirem mensagens internas, já que após a verificação talvez a lista de mensagens não tenha sido esvaziada. Assim, as mensagens de entrada são adicionadas às que já existem.

A idéia geral dos CFSs é começar a partir de uma *tabula rasa* (sem nenhum conhecimento) usando um gerador de população randômica e deixar o programa aprender por indução.

Vamos olhar agora o que é necessário para ser um ANIMAT (animal + robot), por exemplo um sapo: Vejamos o comportamento emitido por Kermit, o sapo. Ele vive em um pântano digital e se move randomicamente por ele. Quando um objeto voador pequeno aparece, e não tem listas, Kermit come-o, porque provavelmente trata-se de uma mosca ou outro inseto voador. Se ele tem listas, será melhor deixar o inseto sossegado, porque Kermit não quer se meter com abelhas, vespas ou zangões.

Se Kermit encontra um objeto grande se movendo, ele imediatamente usa seus efetores para cair fora tão rápido quanto possível.

Parte destes padrões de comportamento pode ser expresso como um conjunto de condição-ação, como segue:

1 IF small, flying object to the left THEN send @

2 IF small, flying object to the right THEN send %

3 IF small, flying object centered THEN send $

4 IF large, looming object THEN send !

5 IF no large, looming object THEN send *

6 IF * and @THEN move head 15 degrees left

7 IF * and % THEN move head 15 degrees right

8 IF * and $ THEN move in direction head pointing

9 IF ! THEN move rapidly away from direction head pointing

Agora, este conjunto de regras pode ser codificado dentro de um sistema classificador. Uma possível codificação pode ser, usando $\sim$ para negação.

 ©88-09, Pedro Kantek

IF	THEN	explicação
0000 00 00	00 00	O primeiro 0000 significa small, o primeiro 00 significa flying, o segundo 00 significa left. Na ação, 00 00 significa enviar um @(regra 1)
0000 00 01	00 01	o segundo 01 significa chegada pela direita. A saída 00 01 é enviar um % (regra 2)
0000 00 10	00 10	O segundo 10 significa centered, e a saída 00 10 é enviar um $ (regra 3)
1111 01 ##	11 11	1111 significa large, 01 não voador, # # significa não importa e a saída 11 11 é enviar ! (regra 4)
∼1111 01 ##	10 00	a saída 10 00 é enviar um * (regra 5)
1000 0000	01 00	Se chega um * e um @, a saída é 01 00 (girar 15 graus à esquerda, que é a regra 6
1000 0001	01 01	Se chega um * e um %, a saída é 01 01 (girar 15 graus à direita, regra 7
1000 00 10	01 10	Se chega um * e um $, a saída é andar um passo em frente (regra 8)
1111 ## ##	01 11	Se chega um !, corra (regra 9)

A seguir, um algoritmo para um sistema CFS sem aprendizado.

```
1: CFS sem aprendizado
2: inicio: t ← 0
3: Lista de mensagens (ML) inicialmente vazia
4: População de classificadores randomicamente inicializada
5: enquanto teste de fim (tempo, fitness, etc) faça
6:    t ← t + 1;
7:    detetores identificam quando há mensagens de entrada (que vão para ML(t))
8:    compara ML(t) com os classificadores e os que batem vão para ML'(t)
9:    ML(t) recebe as novas mensagens mandadas pelos efetores acionados por ML'(t)
10: fimenquanto
```

Algoritmo genético

O algoritmo da brigada de incêndio provê um bom procedimento para avaliar regras e decidir entre alternativas competidoras. Mas ainda precisamos achar uma maneira de injetar regras, possivelmente melhores, no sistema. As regras de um algoritmo genético convencional serão introduzidas e processadas pelos mecanismos de leilão, pagamento e reforço para avaliar apropriadamente seu papel no sistema. Devemos esquecer de trocar a população toda e olhar mais atentamente para quem substitui quem. Nas aplicações de machine learning não é conveniente a substutuição de toda a população. Aqui precisamos de bom desempenho em performance on-line, enquanto na pesquisa e otimização precisamos convergência (ou perfomance off-line).

De Jong implementou o fator C (de troca generacional). Aqui definiremos uma quantidade chamada proporção de seleção proporção, que indicará quantos elementos substituir. Também definiremos uma quantidade chamada período de AG (TGA) que indicará a quantidade de ciclos de tempo (ciclos de regras e mensagens) que deverão ser invocados entre as chamadas ao AG. Este valor pode ser tratado deterministicamente (AG será chamado a cada TGA ciclos) ou estocasticamente (AG é chamado probabilisticamente com período médio TGA). Além disso a invocação do AG pode ser disparada por determinados eventos (perda de match; baixa performance, etc.).

A seleção por roda da roleta usará como parâmetro o vigor S de cada classificador. É preciso estudar quem sai, e o procedimento de *crowdind* de DeJong (75) pode ser usado.

A mutação deve ser modificada, porque os classificadores usam um alfabeto ternário.

A probabilidade de mutação é usada como anteriormente definida. Mas uma vez atingida um caractere será substituído por qualquer um dos outros dois com igual probabilidade. Com essas trocas, o algoritmo genético segue igual ao tradicional.

Conclusão

Em termos gerais: "Há muita pesquisa ainda por fazer em CFS". Nenhum outro ramo da Computação Evolutiva oferece mais espaço para explorar. Se você está interessado em fazer um PhD no campo, eventualmente deveria dar uma olhada em CFSs. Entretanto, CUIDADO! Nas palavras de Goldberg *"classifier systems are a quagmire—a glorious, wondrous, and inventing quagmire, but a quagmire nonetheless."*.

Outro ramo descendente dos algoritmos genéticos é o da programação genética. Aqui, os indivíduos da população não são seqüências de bits, mas sim programas de computador, armazenados na forma de árvores sintáticas. Tais programas é que são os candidatos à solução do problema proposto. Programação genética não usa o operador mutação e a recombinação se dá pela troca de sub-árvores entre dois indivíduos candidatos à solução. A seguir tem-se a programação evolutiva, na qual se busca predizer o comportamento de máquinas de estado finitas. Apenas dois operadores foram originalmente usados: a seleção e a mutação. As idéias datam de 1966, e não foram muito consideradas na comunidade na computação evolutiva por rejeitar o papel fundamental da recombinação, embora esta abordagem esteja sendo reavaliada. Finalmente, completa a árvore a estratégia evolutiva, proposta em meados dos anos 60, na Alemanha. A ênfase aqui é na auto-adaptação. O papel da recombinação é aceito, mas como operador secundário. Segundo Goldberg, "a estratégia evolutiva teve devotos em certos círculos científicos e de engenheiros, particularmente na Alemanha.

Em 1990, a comunidade da computação evolutiva reuniu esforços para a realização do primeiro evento envolvendo todas as iniciatias. Tratou-se do *Workshop on Parallel Problem Solving from Nature*, que ocorreu em Dortmund. Este congresso foi sucessivamente realizado, e diversos outros surgiram. Uma medida do interesse crescente deste ramo de estudo e do seu rápido desenvolvimento pode ser observado na newsletter eletrônica Genetic Algorithms Digest, de 17 de fevereiro de 99, (Volume 13 : Issue 4), apresenta nada menos que 32 chamadas de congressos ou eventos científicos diretamente relacionados ao tema.

Note-se que a despeito de suas origens bastante diversas, é impossível deixar de mencionar quanto todas estas abordagens têm em comum. Desde o modelo filosófico conceitual inicial (a evolução natural), passando pelo conjunto de operadores e principalmente tendo em vista o objetivo final de tais sistemas, que é o da solução de problemas complexos, todos esses ramos têm muito mais semelhanças do que diferenças. Graças a isso é razoável prever que no futuro, haverá uma unificação da teoria, possivelmente mantendo-se cada uma das alternativas exatamente no papel de alternativas de uma abordagem global. De um modo geral, pode-se dizer que os algoritmos evolutivos representam uma abordagem distinta dos métodos tradicionais, no mínimo pelas seguintes razões:

- trabalham com parâmetros codificados e não diretamente sobre os parâmetros;
- trabalham com uma coleção de pontos candidatos a solução e não apenas com um ponto;
- usam uma função objetivo para pontuar os candidatos, prescindindo desta forma, qualquer conhecimento prévio do problema;
- usam regras de transição probabilísticas.

 ©88-09, Pedro Kantek

É uma hipótese a de que a robustez dos métodos evolutivos se deve a estas características. O Schwefel, outro pesquisador diz *"É muitas vezes admirável o fato de que mesmo estabelecendo parâmetros obviamente ruins, não se consegue evitar bons resultados"*. Isto certamente pode ser descrito como robustez. Citando novamente o mesmo autor, pode-se concordar com *"a melhor coisa que se pode dizer sobre os algoritmos evolutivos é que eles apresentam um arcabouço metodológico que é fácil de entender e usar, que pode ser usado na modalidade de caixa preta, ou ser aberto para incorporar receitas novas ou antigas, que incrementem a sofisticação, especialização ou a hibridização. Pode ser usado em situações dinâmicas, onde o objetivo ou as restrições mudam ao longo do tempo, onde o ajuste de parâmetros ou medidas de desempenho são distorcidas por ruído, ou onde o espaço de soluções é irregular, descontínuo, multimodal, fractal ou de alguma maneira não pode ser atacado pelas ferramentas tradicionais, especialmente quando elas necessitam de predições globais a partir de análises da superfície local"*.

6.5 Implementação

Como diz o Fogel, *"Indivíduos e espécies podem ser vistos como uma dualidade entre seus programas genéticos - chamado genótipo, e sua expressão em forma de traços de comportamento, o fenótipo."*.

O conceito chave na computação evolutiva, como definido por Holland é o da adaptação (ad aptare, ou tornar-se apto a) o que permite usar a computação evolutiva em grande classe de problemas distintos, e unificar a abordagem de sua solução.

Para essa classe de métodos de solução, uma população de soluções é usualmente gerada de maneira randômica e evolui ao longo das gerações que são simuladas no processo, em direção às soluções de maior valor da função objetivo, por meio de processos de seleção, mutação e recombinação.

Denomina-se, na computação evolutiva, a função objetivo como "aptidão"(em inglês "fitness"). A função objetiva pode ser tão sofisticada quanto se queira (ou possa), mas no caso mais simples é uma simples decisão binária (a solução a é melhor ou pior do que a solução b ?). O processo de seleção busca indivíduos com alta aptidão para formar descendência, à qual são aplicados os outros dois processos. O primeiro, indica uma possível modificação aleatória na solução, enquanto o segundo significa a troca de partes da solução entre dois ancestrais gerando dois descendentes.

O conjunto inicial de soluções pode ser aleatório ou pode ser obtido a partir das técnicas convencionais que já são dominadas para resolver instâncias mais simples do problema que está sendo tratado. De um lado, usando-se soluções inicialmente aleatórias, pode-se usar sempre o mesmo algoritmo, os mesmos operadores, e socorrer-se da teoria matemática preexistente, como por exemplo, o teorema dos esquemas. Por outro lado, adaptando o conceito de computação evolutiva a um problema específico e empregando soluções iniciais obteníveis por métodos convencionais, há um trabalho mais pesado ao ser necessário adaptar os operadores usuais da computação evolutiva para o problema específico, mas em compensação, na pior das hipóteses, a solução encontrada é igual à melhor solução obtida anteriormente pelas técnicas convencionais. Qualquer melhoria que venha a ser obtida pelo uso da computação evolutiva representará benefício. Além da função de avaliação, é necessário encontrar uma maneira de codificar as soluções para o problema que se quer resolver. O resultado dessa codificação fará o papel dos cromossomos na evolução natural. A partir desses cromossomos é que a função objetivo deve ser capaz de se manifestar e registrar quão boa é aquela solução. Denomina-se genótipo a este conjunto de cromossomos.

Para gerar novas soluções, usam-se os operadores evolutivos, que se aplicam a um indivíduo (solução) ou a pares delas. Tudo se dá como houvesse uma descendência monoparental, chamada assexuada , na natureza, ou cruzamento entre duas soluções, gerando

descendência, denominada sexuada , na natureza. Tal processo repetido inúmeras vezes simula a passagem de gerações e gerações de seres vivos no mundo biológico. Como o processo está apenas sendo simulado em um computador digital, o fator tempo pode ser comprimido sem perda de qualidade.

Estabelecido um conjunto de soluções – que passarão a funcionar como ascendentes as novas soluções ou os descendentes, obtidas a partir daquelas, sofrem a ação dos chamados operadores evolutivos. Mediante estes operadores, os descendentes passarão a ser diferentes dos ascendentes. Se eles forem melhores, assim julgados pela função de avaliação, terão uma descendência maior, do que se eles forem julgados pouco aptos. A troca de material genético, chamada de recombinação, implica que um par de ascendentes dará origem a um par de descendentes, e cada descendente herdará partes aleatoriamente escolhidas de cada ascendente. A mutação significa a mudança também aleatória de uma parte da solução. No caso mais simples de cromossomos codificados em binário, a mutação seria a simples inversão de um bit. Tanto a recombinação quanto a mutação tendem a ocorrer segundo probabilidades dadas que são parâmetros da técnica.

6.5.1 Técnicas de Otimização

A solução de um problema, como posto aqui deverá ser entendida como a busca da otimização do sistema sob análise. Ou seja, a procura de uma solução, preferencialmente uma boa solução e idealmente a melhor solução. Os problemas estudados são os mais diversos e variados, compartilhando uma única característica: a de não serem triviais ou terem fácil solução. A não existência de método universal de otimização fez com que surgissem inúmeros procedimentos, cada um tendo aplicação limitada apenas a casos especiais. Boa parte das técnicas de otimização, possuem um escopo reduzido a uma classe de problemas, e em geral, quanto mais complexa é a solução, mais especializada e conseqüentemente menos geral é o método de solução, quando este existe.

A seguir, uma rápida visão dos principais métodos existentes:

Baseados em cálculo

A solução é encontrada pela resolução do sistema de equações que descrevem o fenômeno em estudo. Quando tal sistema existe e é solúvel, este é o método ideal por excelência, já que é rápido, simples e exato.

Pesquisa randômica

Pontos no espaço de pesquisa são aleatoriamente gerados, e para cada um a função que está sendo otimizada é calculada. Ao final da execução, o ponto que tiver obtido melhor desempenho é selecionado como solução. Para problemas de menor porte, este processo poderia ser aplicado a todos os pontos do espaço. Nesse caso, o método se denomina pesquisa enumerada, e também nesse caso a solução encontrada é a melhor existente, a despeito do considerável esforço computacional empregado. Este método também é conhecido como ”força bruta”, e raramente empregado.

Métodos do gradiente

Uma classe de métodos de solução foi desenvolvida com base no uso do gradiente da função objetivo em um ponto considerado para guiar a direção de busca. Em linhas gerais, um ou vários pontos são randomicamente gerados, e iniciam uma iteração na qual o próximo ponto é localizado obedecendo a direção dada pelo gradiente da função no ponto anterior.

 ©88-09, Pedro Kantek

Este método apresenta duas deficiências importantes. Funções que apresentem descontinuidades não podem ser usadas (pela impossibilidade do cálculo das derivadas necessárias para obter o gradiente) e funções multimodais, pela possibilidade do método se fixar em um máximo local. Estes métodos são conhecidos genericamente pelo nome de "hill climbing ", ou em português, "subida da encosta".

Pesquisa iterativa

É uma combinação dos dois métodos acima. O método do gradiente é usado para encontrar um máximo. Em vez de encerrar a pesquisa, um novo ponto é gerado (de maneira randômica) e uma nova pesquisa de gradiente é iniciada. Ao final, o ponto com maior valor da função objetivo é a solução. Esse método pode funcionar bem para funções que tenham pontos de máximo da função objetivo rodeados por pontos onde tal função tem valor mínimo. Este tipo de função é difícil de otimizar por qualquer método e aqui a simplicidade da busca interativa geralmente wins the day.

Têmpera simulada

Também conhecida como *simulated annealing*. Variação do método da subida da encosta, na qual no início do processo são feitos movimentos que podem piorar o resultado numérico da função objetivo. A idéia é explorar todo o espaço do problema logo no início a fim de ficar independente do estado inicial.

Até pelo nome, este método baseia-se no processo de resfriamento lento de materiais fundidos. Estes são aquecidos a altos níveis de temperatura, e depois gradualmente resfriados até o estado sólido. O objetivo é produzir um estado com mínimo de energia. Em geral os materiais diminuem sua temperatura, mas existe certa probabilidade de ocorrer uma transição a um estado de maior energia. Esta probabilidade é fornecida por $p = e^{-(\frac{\delta E}{kT}}$ onde δE é a mudança positiva no nível de energia, T é a temperatura e k é a constante de Boltzmann. Assim, no resfriamento, a probabilidade de um grande movimento ascendente é menor do que a probabilidade de um movimento pequeno. A probabilidade do movimento ascendente diminui a medida em que diminui a temperatura.

O processo todo é dependente de como T diminui. Se T diminuir rapidamente formar-se-ão locais estáveis de alta energia (o que pode ser considerado como um mínimo local na linguagem de otimização de funções). Se o tempo em que T varia for maior, provavelmente será encontrado um mínimo global.

Transportando a analogia para a computação, δE não representa mais a mudança de energia mas a mudança no valor da função objetivo, seja ela qual for. No mundo computacional k não tem sentido. Então escolhe-se E e T para que trabalhem bem na forma algorítmica. A formula revisada da probabilidade passa a ser $p' = e^{-(\frac{\delta E}{T}}$

O algoritmo de têmpera simulada é um pouco diferente da subida da montanha simples. A principal diferença é que movimentos que pioram a função objetivo podem ser aceitos. Segundo Fogel, *"técnicas clássicas, como gradiente, subida da montanha determinística e pesquisa randômica pura, têm se mostrado em geral insatisfatórias especialmente quando aplicadas a problemas de otimização não lineares, especialmente aqueles com componentes estocásticos, temporais ou caóticos"*

6.5.2 Formalização algorítmica

Seja $f : \Re^n \rightarrow \Re$ a função objetivo a ser otimizada e $\Phi : I \rightarrow \Re$ a função de aptidão, onde I é o espaço de indivíduos. $\vec{a} \in I$ denota um indivíduo enquanto $\vec{x} \in \Re^n$ indica as coordenadas de um ponto no espaço de soluções. $\mu \geq 1$ denota o tamanho da população de ancestrais, enquanto $\lambda \geq 1$ é o tamanho da população de descendentes, ou seja o

número de indivíduos criados através de recombinação e mutação a cada geração. Se o tempo de vida dos indivíduos está limitado a uma geração, pode-se assumir $\lambda > \mu$. A população na geração t, $P(t) = \{\vec{a}_1(t), ..., \vec{a}_\mu(t)\}$ consiste de indivíduos $\vec{a}_1(t) \in I$. $r_{\Theta_r} : I^\mu \rightarrow I^\lambda$ denota o operador de recombinação, o qual pode ser controlado pelo conjunto de parâmetros sumariados pelo conjunto Θ_r. Similarmente, o operador mutação $m_{\Theta_m} : I^\lambda \rightarrow I^\lambda$ modifica a descendência controlado pelos parâmetros Θ_m. A seleção $S_{\Theta_s} : (I^\lambda \cup I^{\mu+\lambda}) \rightarrow I^\mu$ é aplicada na escolha da população de ascendentes para a próxima geração. Durante a etapa de avaliação, a função de aptidão $\Phi : I \rightarrow \Re$ é calculada para todos os elementos da população e $\iota : I^\mu \rightarrow \{verdadeiro, falso\}$ é o critério de fim. Finalmente, $Q \in \{\phi, P(t)\}$ é o conjunto de indivíduos que são tomados adicionalmente durante a etapa de seleção. Se Q é vazio, os indivíduos de uma geração não são usados como candidatos a ascendentes na próxima geração. Se Q for igual a $P(t)$, o contrário ocorre. Com essa notação, pode-se descrever o algoritmo básico da computação evolutiva como

Algoritmo básico da Computação Evolutiva
$t \leftarrow 0$
inicialização: $P(0) \leftarrow \{\vec{a}_1(0), ..., \vec{a}_\mu(0)\} \in I^\mu$
avaliação: $P(0) : \{\Phi(\vec{a}_1(0)), ..., \Phi(\vec{a}_\mu(0))\}$
enquanto $(\iota(P(t)) \neq verdadeiro)$ **faça**
  recombinação: $P'(t) \leftarrow r_{\Theta_r}(P(t))$
  mutação: $P''(t) \leftarrow m_{\Theta_m}(P'(t))$
  avaliação: $P''(t) : \{\Phi(\vec{a}''_1(t)), ..., \Phi(\vec{a}''_\lambda(t))\}$
  seleção: $P(t+1) \leftarrow s_{\Theta_s}(P''(t) \cup Q)$
  $t++$
fimenquanto

6.5.3 Formalização matemática

Um algoritmo evolutivo genérico é definido como uma 8-upla: $AE = (I, \Phi, \Omega, \Psi, s, \iota, \mu, \lambda)$, onde $I = A_x \times A_s$ é o espaço de indivíduos eA_x, A_s denotam conjuntos arbitrários. $\Phi : I \rightarrow \Re$ denota a função de aptidão (fitness) que assinala valores reais aos indivíduos. $\Omega = \{\omega_{\Theta 1}, ..., \omega\Theta_z \mid \omega_{\Theta i} : I^\lambda \rightarrow I^\lambda\} \cup \{\omega_{\Theta 0} : I^\mu \rightarrow I^\lambda\}$ é um conjunto de operadores genéticos probabilísticos $\omega_{\Theta i}$ cada um dos quais é controlado por parâmetros específicos sumarizados pelos conjuntos $\Theta i \subset \Re$. $s_{\Theta_s} : (I^\lambda \cup I^{\mu+\lambda}) \rightarrow i^\mu$ denota o operador seleção, o qual pode modificar o número de indivíduos de λ ou $\lambda+\mu$ até μ, onde μ e $\lambda \in$ (e $\mu = \lambda$ é permitido. Um conjunto adicional Θ_s de parâmetros pode ser usado pelo operador de seleção. μ é o número de indivíduos ancestrais e λ é o número de indivíduos descendentes. $\iota I^\mu \rightarrow \{verdadeiro, falso\}$ é um critério para o fim do algoritmo e a função de transição de gerações $\Psi : I^\mu \rightarrow I^\mu$ descreve o processo completo de transformação de uma população P em sua geração subseqüente pela aplicação dos operadores genéticos e de seleção
$\Psi = s \circ \omega_{\Theta i1} \circ ... \circ \omega_{\Theta ij} \circ \omega_{\Theta 0}$
$\Psi(P) = s_{\Theta s}(Q \cup \omega_{\Theta i1}(...(\omega_{\Theta ij}(\omega_{\Theta 0}(P)))...))$
$\{i_1, ..., i_j\} \subseteq \{1, ..., z\}$
$Q \in \{0, P\}$

Teorema Fundamental dos Algoritmos Genéticos Embora válido apenas para representações binárias de indivíduos, esta é talvez a primeira e mais importante formalização da garantia de funcionamento dos algoritmos genéticos.

Seja inicialmente, um alfabeto binário $V_1 = \{0, 1\}$. Para efeito deste desenvolvimento

 ©88-09, Pedro Kantek

necessita-se descrever o conceito de esquema, que vem a ser uma máscara, composta de três caracteres, quais sejam o 0, o 1 e um terceiro denominado "coringa"e representado por um *, que deve ser lido como "tanto faz". Portanto, para representar esquemas, necessita-se de um alfabeto trinário $V_2 = \{0, 1, *\}$.

A finalidade do esquema é estabelecer cadeias e questionar se um indivíduo pertence ou não aquele esquema (cadeia). Por exemplo: Seja o esquema `**10`. Pertencerão a este esquema os indivíduos `0010`, `0110`, `1010` e `1110`. Não importa o que aparece nas duas primeiras posições, desde que nas duas últimas apareça `10`. Esta é a leitura da cadeia `**10`.

Se o esquema não possui nenhum `*`, apenas um indivíduo (se existente) pertencerá a esse esquema. Se possui um único `*`, dois indivíduos poderão pertencer a ele, e assim por diante. Posto assim, para um esquema de n asteriscos, o número de indivíduos que podem a ele pertencer é de 2^n. Por outro lado, para indivíduos de tamanho k, podem existir 3^k esquemas diferentes. Seja um conjunto de indivíduos de comprimento 2. Pela regra acima devem existir 9 esquemas, a saber: `00`, `01`, `10`, `11`, `*0`, `*1`, `0*`, `1*` e `**`. Em uma população de n indivíduos há sempre $n \times 2^m$ esquemas contidos na população, já que cada um deles é representativo de 2^m esquemas.

Alguns esquemas são mais genéricos que outros. Para qualificá-los surgem os conceitos de ordem e comprimento de esquema. A ordem de um esquema H, denotada o(H) é o número de posições fixas (ou seja, de uns e zeros) do esquema.

Por exemplo: se H_1=`1***0`, $o(H_1) = 2$, enquanto se H_2 = `******` , $o(H_2) = 0$. O comprimento de um esquema H, denotado por $\delta(H)$ é a distância entre o primeiro e o último zero ou um. Se H_3 =`10***1**`, $\delta(H_3) = 6 - 1 = 5$

Com estes conceitos, pode-se analisar o efeito individual e coletivo dos operadores seleção, crossover e mutação sobre os esquemas contidos em uma população.

Analisando primeiro o efeito sobre a seleção (reprodução). Escreve-se $m(H, t)$ como sendo o número m de exemplares de H presentes em uma população no instante t. Lembre-se de que há mais esquemas que indivíduos, vários esquemas estão representados por um único indivíduo, o que caracteriza o paralelismo de funcionamento do esquema global. A probabilidade de um indivíduo i ser selecionado para reprodução é

$$p_i = \frac{f_i}{\sum f_i}$$

onde f_i é a aptidão do indivíduo i e $\sum f_i$ é a somatória de todas as aptidões da população. A partir deste conceito, pode-se estabelecer a quantidade esperada de esquemas na geração seguinte a uma dada, o que seria $m(H, t + 1)$, e tem-se

$$m(H, t + 1) = m(H, t).n.f(H)/\sum f_i$$

onde $m(H, t)$ é a quantidade de indivíduos que pertencem ao esquema H na geração t, n é a quantidade de indivíduos na população, e $f(H)$ é a aptidão média dos indivíduos que pertencem ao esquema H. Mas, lembrando que a aptidão média de toda a população $\bar{f}$ pode ser escrita como $\bar{f} = \sum \frac{f}{n}$ e pode-se rescrever o processo de crescimento via reprodução de um esquema como segue:

$$m(H, t + 1) = m(H, t)\frac{f(H)}{\bar{f}}$$

A leitura desta fórmula é: a quantidade de indivíduos que pertencem ao esquema H na próxima geração é igual a quantidade de indivíduos que pertencem ao esquema H na geração anterior, multiplicado pela aptidão média dos indivíduos que pertencem ao esquema H, dividido pela aptidão média da população. Desta equação pode ser inferido o fato de que esquemas com aptidão maior do que a média da população, tendem a

crescer ao longo do tempo, enquanto que esquemas com aptidão menor que a média tendem a diminuir.

Segue-se a análise do efeito do operador de crossover sobre a sobrevivência dos esquemas. Supondo um crossover de 1 ponto e um esquema de comprimento ((H) = k, sua probabilidade de se rompido é de

$$p_r = \frac{\delta(H)}{k-1}$$

Um esquema sobrevive quando o corte da recombinação ocorre fora do seu comprimento definido e a probabilidade seria

$$p_s = 1 - \frac{\delta(H)}{k-1}$$

Já que o própria recombinação ocorre com uma dada probabilidade, diga-se pc, tem-se que a probabilidade de sobrevivência de um esquema H, a uma operação de crossover com probabilidade pc de ocorrer é

$$p_s \geq 1 - p_c.\frac{\delta(H)}{k-1}$$

Juntando as duas considerações (seleção e crossover), tem-se a seguinte probabilidade de sobrevivência de um esquema

$$m(H, t+1) \geq m(H, t).\frac{f(H)}{\bar{f}}[1 - p_c.\frac{\delta(H)}{k01}]$$

Olhando esta fórmula, percebe-se que H cresce (ou diminui) dependendo de dois fatores conjugados: de que ela tenha aptidão maior (menor) que a média e pequeno (grande) comprimento. Note-se que houve troca do sinal = pelo sinal $\geq$ uma vez que embora a recombinação possa ocorrer, nada impede que o resultado obtido seja exatamente igual ao que havia antes da recombinação. Desta maneira, a probabilidade do esquema sobreviver pode ser maior ainda do que a calculada pela fórmula, daí o sinal de $\geq$.

Finalmente, vale considerar o efeito do operador de mutação. Um esquema sobrevive desde que todos os seus componentes não sejam modificados. Supondo uma mutação com probabilidade p_m, um alelo sobreviverá com probabilidade $(1 - p_m)$. A probabilidade total de sobrevivência do esquema é obtida multiplicando-se $(1 - p_m)$ por si mesmo tantas vezes quantas for a ordem do esquema. Ou seja,

$$p_{sm} = (1 - p_m)^{o(H)}$$

Entretanto, como pm sempre tem valores pequenos, a probabilidade de sobrevivência devida a mutação pode ser escrita como (Goldberg, 1989)

$$p_{sm} = 1 - o(H).p_m$$

Finalmente, juntando os 3 fatores que intervém na sobrevivência de um esquema, tem-se

$$m(H, t+1) \geq m(H, t).\frac{f(H)}{\bar{f}}[1 - p_c.\frac{\delta(H)}{k-1} - o(H).p_m]$$

E aqui chega-se à formulação matemática do que é conhecido como o teorema fundamental dos algoritmos genéticos: esquemas curtos, de baixa ordem, de aptidão maior do que a média, surgem nas gerações subsequentes com número exponencialmente crescente. Tais esquemas com as características acima, recebem o nome de building blocks, ou blocos construtivos das soluções.

 ©88-09, Pedro Kantek

Mundo natural	Algoritmos genéticos
cromossomo	seqüência
gene	característica, caractere ou detetor
alelo	valor da característica
locus	posição da seqüência
genótipo	estrutura
fenótipo	conjunto de parâmetros, alternativa de solução, estrutura decodificada
epístase	não linearidade (ou influência de um gene sobre outros genes)

Figura 6.2: Comparação entre a genética natural e os algoritmos genéticos

Algoritmo básico Uma visão em alto nível do que seja um algoritmo genético.

1. Inicializar a população de cromossomos (soluções).
2. Avaliar cada cromossomo da população.
3. Criar novos cromossomos a partir da população atual, aplicar mutação e recombinação, substituindo os ascendentes pelos descendentes.
4. Se o critério de fim foi alcançado, deve-se terminar. Caso contrário, retorna-se à etapa 1.

6.5.4 Terminologia

A terminologia usada nos algoritmos genéticos é estreitamente relacionada ao universo genético natural, como pode-se ver na tabela 6.2

6.5.5 Idéias Comuns à CE

A inspiração básica para os posteriores desenvolvimentos aqui mostrados, é sempre o mundo natural. Os termos usados são tomados por empréstimo das ciências biológicas, usando-se algum tipo de analogia com o que já existe, embora como tenha dito Melanie Mitchel *"as entidades a que (os termos) se referem são muito mais simples do que aquelas do mundo real"*. Por exemplo, quando se fala de recombinação no âmbito da CE, o fenômeno é uma extraordinária simplificação do mecanismo de troca de material genético que existe nos seres vivos da natureza.

Fitness Landscape

O conceito de fitness landscape, também conhecido como Adaptative surface foi originalmente definido pelo biólogo Wright em 1931. É a representação do espaço de todos os possíveis genótipos, cada um com seu fitness. Supondo indivíduos de x cromossomos e com um número real k associado a cada cromossomo a fitness landscape (superfície de aptidão) é uma figura formada em um sistema de x+1 eixos. Nos x eixos representa-se o indivíduo através dos seus diversos valores em cada eixo, correspondendo aos valores de cada cromossomo que o compõe. Finalmente, no eixo que sobra representa-se o seu valor de fitness. Para efeito de clareza e exemplificação, seja um caso de indivíduos binários que tenham 2 cromossomos. Necessita-se portanto de 2 eixos para representar os indivíduos e em cada eixo estarão marcados os alelos possíveis para este cromossomo (eixo): Aqui estão representados os 4 indivíduos possíveis neste universo de cromosso-

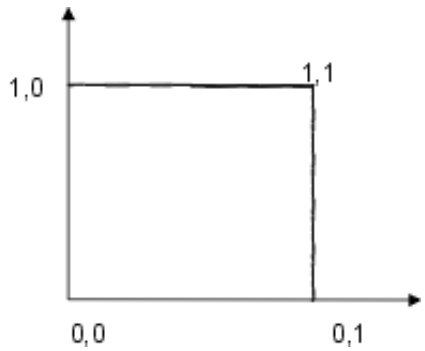


Figura 6.3: Exemplo de fitness landscape para indivíduos com 2 cromossomos

mos binários de comprimento 2: são eles os indivíduos 00, 01, 10 e 11. Supondo que esteja associado a cada um deles um determinado fitness, por hipótese, 0.9, 0.5, 0.4 e 0.0 e representado isso tridimensionalmente, ter-se-ia

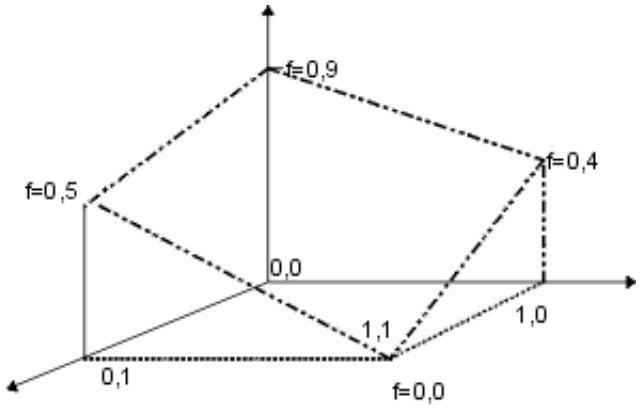


Figura 6.4: Exemplo de fittness landscape para indivíduos com 3 cromossomos

À superfície formada pelos pontos (0,0,0.9); (0,1,0.5); (1,0,0.4) e (1,1,0) dá-se o nome de fitness landscape e ela representa todos os possíveis valores para a função objetivo que se está estudando. O nome landscape (paisagem) vem do fato de para problemas de maior dimensionalidade, ela é formada por montanhas, picos e vales entre outros acidentes geográficos. De acordo com a idéia original de Wright a evolução faz com que as populações se movam sobre essa superfície através de caminhos particulares. Adaptação (no sentido evolutivo) pode ser aqui entendido como o movimento ao redor de picos locais. Para encerrar a analogia, deve-se ressaltar que o landscape não é – em casos gerais – fixo, sendo dependente, principalmente no mundo real, da existência e da co-existência de outros indivíduos que iteragem entre si. Esta idéia foi melhorada por Atmar em 1979, quando este sugeriu que o conceito seria mais facilmente entendido se o plano fosse invertido, e os picos se transformassem em vales. Em vez da busca de alto valores de aptidão, o modelo agora buscaria minimização de erros. A diferença está na ação da gravidade sobre a paisagem. Rápidas descidas são seguidas por períodos de estagnação, principalmente se a paisagem é estática.

 ©88-09, Pedro Kantek

Desafio

- Localize o GALOPPS na internet e baixe-o no seu computador. Implemente um problema simples.

6.6 CE: mais conceitos

6.6.1 Hibridização

A técnica de hibridização, resulta na junção de uma boa maneira convencional de resolver um problema aos conceitos usuais da computação evolutiva. O resultado costuma ser melhor que o obtido com qualquer uma das duas técnicas utilizadas isoladamente. Como diz Davis "hibridizando um algoritmo genético com um algoritmo corrente, pode-se produzir um algoritmo que é melhor do que os dois anteriores"(Davis, 1991). A hibridização agrega a representação de dados usual no domínio original, bem como as técnicas de otimização já porventura existentes. Isto permite a incorporação de heurísticas otimizadoras ao conjunto de operadores genéticos (recombinação e mutação) que passam portanto a ser dependentes do domínio. Nesse sentido, a computação evolutiva passa a ser muito mais uma filosofia de otimização do que um pacote de software pronto para usar.

Um exemplo de hibridização possível é quando o problema exige codificação com base em números reais e não em números binários. Alguns conceitos teriam que ser adaptados: por exemplo, a mutação não seria mais a troca simples de um bit, mas a geração de um novo real, possivelmente dentro de um intervalo dado. Já a recombinação de dois reais, poderia ser qualquer número compreendido entre eles, ou talvez a sua média.

Outra possibilidade de hibridização é quando o problema envolve algoritmos de ordem, como por exemplo, no célebre problema do caixeiro viajante (Um viajante precisa passar por um certo número de cidades, que estão ligadas por uma determinada malha viária. Associado a cada trecho do caminho, há uma função de penalização: custo, tempo, desgaste, entre outros. O objetivo do problema é resolver qual o trajeto que minimiza esta penalização). Ou no de colorir um grafo, que pode ser assim descrito: dado um grafo com certo peso em cada nodo e dadas n cores, o problema busca encontrar a mais alta soma de pesos de nodos pintados com alguma cor. Pode ser colorido qualquer nodo com qualquer cor, desde que nenhum par de nodos conectados tenham cor igual. O algoritmo convencional para este problema (chamado de guloso) sugere a seguinte estratégia:

a) classificar o conjunto de nodos em ordem decrescente de peso

b) aplicar a cada nodo a primeira cor legítima que ele puder ter.

A crítica a esta solução é que ela sempre procede localmente. Não existe modo de encontrar uma solução global satisfatória, já que o problema é NP-completo. Para usar computação evolutiva, a questão é: como hibridizar este problema e a solução local? É necessário criar uma função de avaliação de um dado cromossomo, que agora representará seqüência de nodos de uma determinada cor. O operador de mutação pode ser o embaralhamento de uma sub-lista dentro do cromossomo. O operador de recombinação, pode permutar itens entre os dois ascendentes, associado a alguma função que impeça de gerar soluções inválidas. Em Davis, onde este problema está citado, há a descrição de um problema real (100 nodos e 3 cores) onde o algoritmo guloso obteve como resultado 9.590 pontos, o melhor resultado em 4000 gerações aleatórias obteve 9.610 pontos e um algoritmo genético hibridizado obteve 10.594 pontos.

6.6.2 Roda da roleta

Trata-se de um operador de seleção cujo propósito é obter maiores chances de reprodução no todo para aqueles membros da população de que tenham melhor aptidão. Segue um algoritmo semelhante a:

1. Somar as aptidões de todos os membros da população. Chamar o resultado de aptidão total
2. Gerar um número randômico n, entre 0 e a aptidão total
3. Retornar o primeiro membro da população cuja aptidão, somada as aptidões anteriores seja maior ou igual a n.

Ainda que o procedimento de seleção seja randômico, cada ascendente tem uma chance de ser selecionado que é proporcional à sua aptidão. Tendendo ao limite, após um certo número de gerações, o algoritmo desprezará ascendentes com baixa aptidão e acabará valorizando aqueles com alta aptidão.

Número do cromossomo	1	2	3	4	5	6
Aptidão	11	7	1	21	3	16
Aptidão acumulada	11	18	19	40	43	59

Sejam agora a geração dos seguintes números aleatórios entre 0 e 59

Número aleatório gerado	36	8	27	51
Cromossomo selecionado	4	1	4	6

Neste exemplo, ao se gerar um aleatório entre 0 e 59, obteve-se 36. Percorrendo a tabela anterior, percebe-se corresponder este aleatório ao cromossomo nº 4 (já que 36 é maior que 19 e menor que 40). Identicamente para os números 8, 27 e 51.

6.6.3 Mutação de bit

É um operador evolutivo, conhecido como mutação. Trata-se da inversão aleatória de um determinado bit. Este operador ocorre com uma dada probabilidade, originalmente estabelecida em um valor baixo (por exemplo 0.01, significando 10 chances em 1000, ou 1%). Dada esta taxa T, diz-se que um bit tem probabilidade T de ser randomicamente substituído. Um exemplo na figura 6.5 ao final desta aula.

No exemplo da figura **??** dados os 3 cromossomos (denominados ascendentes na tabela acima), e dada uma taxa de mutação de 0.01, para cada bit dos cromossomos é gerado um aleatório. Sempre que o número gerado for superior a 0.01, a mutação não ocorre. Quando, ele for menor (casos do 4º bit do 2º cromossomo e do 3º bit do 3º cromossomo), o operador mutação gera um novo bit. Note-se que em 50% dos casos, o bit gerado é igual ao que havia antes (caso do 3º cromossomo), e nos outros 50% o bit é modificado. Em resumo, para uma taxa de mutação de 0,01, a taxa efetiva de mudança de bits é igual à metade deste valor. Recombinação de 1 ponto Ocorre quando partes de cromossomos ascendentes são invertidos após um ponto randomicamente selecionado criando 2 descendentes. Por exemplo, dado um ascendente

1	1	1	1	1	1

e outro ascendente

0	0	0	0	0	0

poderíamos ter um descendente

1	1	1	1	0	0

e outro descendente

 ©88-09, Pedro Kantek

0	0	0	0	1	1

se o corte tivesse se dado após o quarto bit nos ascendentes. Uma questão importante é que este operador produz descendentes que são radicalmente diferentes de seus ascendentes. Outra, é que este operador não introduz nenhuma diferença nos bits das posições em que ambos os ascendentes tiverem o mesmo valor. Uma instância extrema ocorre quando ambos os ascendentes são completamente iguais. Também o serão os descendentes independente da ocorrência da recombinação de 1 posição.

cromossomo ascendente				números randômicos				novo bit	cromossomo ascendente			
1	0	1	0	.799	.109	.257	.289	-	1	0	1	0
1	1	0	0	.167	.890	.310	**.009**	1	1	1	0	**1**
0	0	1	1	.766	.673	**.003**	.345	1	0	0	1	1

Figura 6.5: Exemplo de uma mutação de bit

6.6.4 Normalização linear

Técnica para melhorar o esquema de alocação de aptidão aos indivíduos da população. Ordena-se os cromossomos por seus valores decrescentes. Cria-se uma aptidão que começa em um valor constante e decresce linearmente. Este valor constante e a taxa de decremento são parâmetros da técnica

Avaliação original	192	11	8	5	2	1
Aptidão é igual a avaliação	192	11	8	5	2	1
Normalização linear com início = 100, decremento = 3	100	97	94	91	88	85
Normalização linear com início = 100, decremento = 20	100	80	60	40	20	0

6.6.5 Janelamento

Técnica para aumentar a aptidão dos indivíduos que têm avaliação muito baixa. Acha-se o valor mínimo da população. Dá-se a cada cromossomo como aptidão o valor em que cada um excede este mínimo. Opcionalmente um valor pequeno acima deste valor mínimo pode ser dado aos piores cromossomos, já que na maneira original eles não teriam chance de reprodução.

Avaliação original	192	11	8	5	2	1
Janelamento com mínimo = 0	191	10	7	4	1	0
Janelamento com mínimo = 5	191	10	7	5	5	5

6.6.6 Recombinação de 2 pontos

Nem sempre o operador de recombinação de 1 ponto consegue combinar certas características codificadas dentro dos cromossomos. Veja-se no exemplo a seguir dois cromossomos, nos quais os caracteres em negrito representam características importantes que devem ser preservadas (o que no linguajar de Holland, seria um esquema).

```
cromossomo 1: 0 1 0 1 1 0 0 1 0 1 1 0 1 1
cromossomo 2: 1 0 0 1 0 1 1 0 1 1 1 1 0 0
```

A recombinação de 1 ponto não consegue juntar estes dois esquemas em um único descendente, já que os pontos positivos do cromossomo 1 estão nas duas extremidades. Não importando onde o ponto seja escolhido, o cromossomo 1 vai se partir e seu esquema não passará à descendência. Uma solução para este problema seria usar o operador de recombinação de 2 pontos. Ele funciona tal como o de 1 ponto, exceto que ele corta em 2 lugares aleatórios e o material é invertido entre os ascendentes nesses 2 pontos. Veja-se a seguir, como isso poderia ser feito no exemplo acima. Aqui, o sinal de = representa o local do corte.

```
Ascendente 1  : 0 1 0 1 = 1 0 0 1 0 1 = 1 0 1 1
Ascendente 2  : 1 0 0 1 = 0 1 1 0 1 1 = 1 1 0 0

Descendente 1: 0 1 0 1   0 1 1 0 1 1   1 0 1 1
Descendente 2: 1 0 0 1   1 0 0 1 0 1   1 1 0 0
```

6.6.7 Recombinação uniforme

Para cromossomos e esquemas mais sofisticados, foi desenvolvida uma recombinação que tem a possibilidade de cortar um cromossomo em até n pontos, onde n é o comprimento do mesmo. Ele funciona como segue: 2 ascendentes são selecionados e 2 descendentes são produzidos. Para cada posição de bit nos 2 descendentes, decide-se randomicamente qual ascendente contribuirá com seu valor de bit para qual descendente.

Veja-se no exemplo

```
Ascendente 1    : 0 0 0 1 0 1 1
Ascendente 2    : 1 1 0 1 1 0 1

Máscara : 1 1 0 1 0 0 1 (funciona como selecionador do ascendente)

Descendente 1  : 0 0 0 1 1 0 1
Descendente 2  : 1 1 0 1 0 1 1
```

A máscara indica qual ascendente cederá seu bit para o descendente 1. A negação da máscara cria o descendente 2. Comparação A tabela a seguir, retirada de (Bäck, 1993 - pág. 19 e Bäck 1996, pág 132) retrata bem as diferenças conceituais sobre as três abordagens:

 ©88-09, Pedro Kantek

Característica	Estratégia Evolutiva	Programação Evolutiva	Algoritmos Genéticos
Representação	números reais	números reais	strings binários
Auto-adaptação	desvio padrão e covariância	Variância	não há
Aptidão	valor da função objetivo	valor da função objetivo após normalização	valor da função objetivo após normalização
Mutação	operador principal	operador único	operador secundário
Recombinação	variantes distintas, importante para a auto-adaptação	não há	operador principal
Seleção	determinística e extintiva	probabilística e extintiva	probabilística e preservativa
Restrições	Restrições, inequações e arbitrário	não há	limites simples, providos pelo mecanismo de codificação
Teoria	Taxa de convergência para casos especiais, (1+1)-ES, (1+λ)-ES, (1,λ)-ES, convergência global para (μ+λ)-ES	Taxa de convergência para casos especiais, (1+1)-EP, convergência global para (1+1)-EP	teoria do processamento de esquemas, convergência global para a versão elitista

6.6.8 Trabalho pedido em 2002

O Objetivo

a) Tornar operacional um engenho de computação evolutiva (sugestões: GALOPPS e/ou TUDEL).

b) Vivenciar as decisões de "customizar"uma aplicação padrão para um caso particular que se quer resolver, verificando as dificuldades dessa tarefa.

c) Finalmente, comprovar o valor da técnica, uma vez que permite resolver um problema que não se sabe resolver.

Lembrar que como toda técnica exploratória e probabilística, não há garantia de chegar a bons resultados.

O Trabalho "Quanto devemos produzir ?"

Escolhi um problema de otimização, diretamente em um livro texto de pesquisa operacional. Como o objetivo deste trabalho é pedagógico, escolhi um problema que pode ser resolvido por outro meio, e consequentemente o será. Ao agir assim, poderemos – ao final – comparar o desempenho das duas técnicas. O problema pode ser ssim descrito, já traduzido:

Uma divisão de uma companhia de plásticos produz 3 produtos: gafacos, pacotes e ofertões. Um gafaco é um utensílio plástico que pode ser usado como garfo, colher ou faca. Um pacote consiste de um gafaco, um guardanapo e um canudinho. O ofertão é um conjunto de 100 pacotes, com um adicional de 10 gafacos.

A produção de 1000 gafacos requer 0.8 horas na máquina de moldagem; 0.2 horas de supervisão e $ 2.50 em custos diretos. A produção de 1000 pacotes (incluindo 1 gafaco, 1 guardanapo e 1 canudinho cada um) requer 1.5 horas da empacotadora, 0.5 horas de supervisão, e $ 4.00 de custos diretos. A produção de 1000 ofertões requer 2.5 horas de empacotadora, 0.5 horas de supervisão, 10 gafacos, 100 pacotes e $ 8.00 em custos diretos.

Qualquer um dos 3 produtos pode ser vendido em quantidades ilimitadas, aos preços de $ 5.00,$ 15.00 e $ 300.00 por milhar, respectivamente. Considerando que há 200 horas produtivas no mês, quais produtos e em que quantidades devem ser produzidos, a fim de se otimizar o lucro ?

Fazendo as tratativas matemáticas:

x_1 = número de gafacos produzidos em milhares (1)

x_2 = número de pacotes produzidos em milhares (2)

x_3 = número de ofertões produzidos em milhares (3)

y_1 = gafacos vendidos como gafacos (4)

y_2 = pacotes vendidos como pacotes (5)

y_3 = ofertões vendidos como ofertões (6)

E relacionando as variáveis x às variáveis y, fica:

$y_1 = x_1 - x_2 - 10x_3$ (7)

$y_2 = x_2 - 100x_3$ (8)

$y_3 = x_3$ (9)

A fórmula (7) diz que serão vendidos tantos gafacos, quantos produzidos subtraindo-se o número de pacotes produzidos e 10 vezes o número de ofertões produzidos. A fórmula (8) diz que serão vendidos tantos pacotes quantos produzidos, substraindo-se 100 vezes o número de ofertões produzidos. Finalmente a (9) diz que serão produzidos tantos ofertões quantos forem vendidos.

Operando-se sobre as fórmulas, para obter as expressões em x, fica:

$x_1 = y_1 + y_2 + 110y_3$ (10)

$x_2 = y_2 + 100y_3$ (11)

$x_3 = y_3$ (12)

A (10) diz que serão produzidos tantos gafacos quantos forem vendidos, mais o número de pacotes vendidos, mais 110 vezes o número de ofertões vendidos. A (11) informa que serão produzidos tantos pacotes quantos forem vendidos mais 100 vezes o número de ofertões vendidos. A (12) simplesmente reafirma o que dizia a (9).

O lucro da operação pode ser descrito como segue:

Lucro $= 5y_1 + 15y_2 + 300y_3 - 2.5x_1 - 4x_2 - 8x_3$ (13)

ou Lucro $= 2.5x_1 + 6x_2 - 1258x_3$ (14)

A (13) indica os preços de venda substraídos dos custos de fabricação. Já a (14) consolida as informações da (13) apenas em função das variáveis de produção.

As restrições do problema são $0.8x_1 \leq 200$ (15) (restrição do tempo de injeção)

$1.5x_2 + 2.5x_3 \leq 200$ (16) (restrição do tempo de empacotamento)

$0.2x_1 + 0.5x_2 + 0.5x_3 \leq 200$ (17) (restrição do tempo de supervisão)

Finalmente, restam as restrições de não negatividade:

$x_1 - x_2 - 10x_3 \geq 0$ (18)

$x_2 - 100x_3 \geq 0$ (19)

$x_3 \geq 0$ (20)

$x_1 \geq 0, a; x_2 \geq 0,\ x_3 \geq 0$ (21)

Implementação no Engenho Evolutivo

Relembrando que no uso de CE é necessário resolver 3 problemas:

- A função que vai valorizar uma determinada solução encontrada

 ©88-09, Pedro Kantek

- O esquema de codificação dos parâmetros da solução dentro dos cromossomos
- Os operadores genéticos a serem usados

Por simplicidade, sugiro usar um esquema de representação binária dos dados. Agregada a esta decisão, acompanha a utilização dos operadores usuais (seleção, crossover e mutação) sem nenhuma modificação. Isso nos permitirá usar qualquer engenho sem nenhuma alteração.

Para a representação dos cromossomos deve-se criar 3 variáveis (x1, x2 e x3) representadas por um número binário de bits, cada uma. O cromossomo terá um comprimento total de bits.

O tratamento que estes alelos deverão sofrer é: ...

A função que vai valorizar as soluções é a própria função objetivo do problema. Como, em geral, a CE trabalha com problemas onde não há restrições, será necessário usar o método de penalidades, como descrito em [Gol89, pág 85]

> "In a penalty method, a constraint problem in otimization is transformed to an unconstrained problem by association a cost or penalty with all constraint violations...here we usually square the violation of the constraint for all violations."

Usando a recomendação de Goldberg, à função objetivo serão agregados (com sinal negativo) os quadrados das violações das restrições. Com isso, a função objetivo tornar-se-á: função objetivo = lucro - penalidades, onde o lucro é o descrito em (14) e as penalidades são:

$P_1 = (250 - x_1)^2$, se $x_1 > 250$ (22)

$P_2 = (200 - (1.5x_2 + 2.5x_3))^2$, se $1.5x_1 + 2.5x_2 > 200$ (23)

$P_3 = (200 - (0.2x_1 + 0.5x_2 + 0.5x_3))^2$, se $0.2x_1 + 0.5x_2 + 0.5x_3 > 200$ (24)

$P_4 = (x_1 + x_2 + 10x_3)^2$, se $x_1 + x_2 + 10x_3 < 0$ (25)

$P_5 = (x_2 - 100x_3)^2$, se $x_2 - 100x_3 < 0$ (26)

$P_6 = x_3^2$, se $x_3 < 0$ (27)

Note que P_6 pode ser desconsiderada, se o alelo escolhido não tiver como representar variáveis negativas.

$P = P_1 + P_2 + P_3 + P_4 + P_5 + P_6$ (28)

Agora a nova função objetivo é $FO = L - P$ (29)

Como já se mencionou, a técnica da CE não é determinística, e portanto não é correto executar uma única vez o programa. A resposta encontrada, pode nunca mais se reproduzir, o que inviabilizaria o uso deste método. Para fugir de (ou minimizar) este problema, costuma-se usar uma população de soluções, ao invés de uma só. Faz-se isto, rodando diversas vezes o mesmo programa, com populações iniciais aleatórias e consequentemente diferentes a cada execução. Assim, para considerar um resultado, o mesmo terá que ser rodado pelo menos 5 vezes e apresentada a média (ou o melhor) das execuções.

Decisões de implementação:

Número de rodadas:

Número de indivíduos por rodada:

Tamanho do cromossomo:

Número de gerações em cada rodada:

Método de seleção:

Método de match:

Valor inicial de normalização linear (se aplicável):

Decremento da normalização: (se aplicável):

Parâmetro de elitismo: cromossomos

Taxa de mutação:

Taxa de crossover:

Tipo de crossover:

Quantidade de pontos de crossover:

As diferenças entre o método simplex e o método usando algoritmos genéticos podem assim ser estabelecidas:

	x1	x2	x3	Valor da f.o.
Usando simplex				
Usando a.g.				
Diferenças absolutas				
Diferença relativa				

As Regras

O trabalho pode ser feito por equipes de até 3 pessoas. Admite-se (e encoraja-se) o trabalho individual, desde que seja de interesse de algum aluno.

Cada equipe deve construir um software INÉDITO.

As turmas se constituirão oficialmente pela informação ao professor de sua composição no primeiro dia de laboratório

O prazo para construção do software é de 6 sessões de laboratório, a ocorrerem nos dias 9, 16, 23, 30 de outubro e 6 e 13 de novembro de 2002.

O limite para entrega do relatório sobre a solução do problema, junto com o fonte, o executável e os arquivos de parâmetros é o do final da aula do último dia.

Prova de autoria: Fica estabelecido a pertinência da realização de quaisquer provas de autoria, para qualquer aluno em quaisquer momento do trabalho. A prova poderá ser dispensada, a critério do professor, desde que ocorra um acompanhamento sistemático do desenvolvimento do projeto.

Critérios de pontuação:

Os trabalhos serão pontuados segundo o seguinte critério:

Trabalho não entregue: grau E

Trabalho entregue incompleto, com falhas graves de funcionamento: D

Trabalho com falhas leves: C

Trabalho sem falhas: B

Trabalho perfeito com adicionais de ergonomia, segurança, apresentação, diversas alternativas para efeito de comparação do método ou outros adicionais, e principalmente, FUNCIONANDO: A Os graus acima terão a seguinte conversão para numérico, para efeito de composição da nota bimestral: A=10, B=8, C=6, D=4, E=2.

Para o terceiro bimestre, a média será composta pela avaliação deste trabalho, com 80% e pelos exercícios individuais valendo 20%.

Dúvidas ? kantek @pr.gov.br

Resposta

Adotando-se o algoritmo simplex para solução deste problema, obtem-se os seguintes valores

a solução é $y_1 = 116.7$, $y_2 = 133.3$, e $y_3 = 0$, com um lucro de $L = 1425\$$

os valores para x_1, x_2 e x_3 são

$x_1 = 250$

$x_2 = 133.3$

$x_3 = 0$

que correspondem a um lucro de $ 1.425 unidades de capital.

 ©88-09, Pedro Kantek

Desafio

- Tente esquematizar como seriam funções escritas em C que fizessem a seleção, mutação e recombinação. Estude uma estrutura de dados adequada para representar a população sob estudo.

6.7 Programação Genética

6.7.1 Outros ramos da Computação Evolutiva

Na figura a seguir, descrevem-se os principais ramos da Computação Evolutiva, que é mais ampla do que os algoritmos genéticos, já vistos. Cada uma das folhas desta árvore

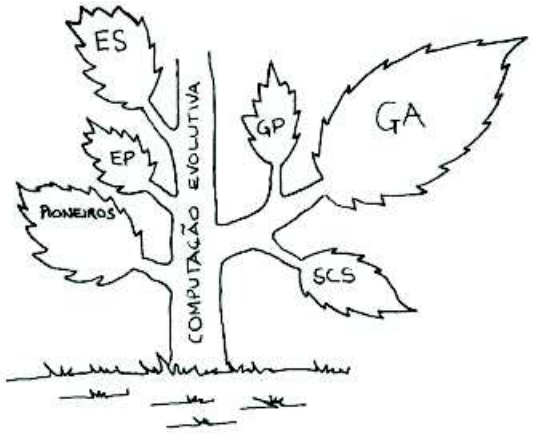


Figura 6.6: Uma distribuição dos ramos da CE

representa uma tendência ou uma concepção prévia a respeito de algumas características que a simulação da evolução natural deve ter. Historicamente, as primeiras iniciativas na área (representadas pela folha chamada de "pioneiros"), são de biólogos e geneticistas interessados em simular os processos vitais em computador, o que recebeu na época o nome de "processos genéticos". Alguns desses cientistas citados em Goldberg (1989) são: Barricelli, 1957, 1962; Fraser, 1960, 1962; Martin e Cockerham, 1960. Um biólogo, Rosenberg, em 1967 ao escrever sua tese de doutorado, simulou uma população de seres unicelulares, estrutura genética clássica (um gene, uma enzima), com estrutura diplóide, com cromossomos de 20 genes e 16 alelos permitidos em cada um (Goldberg, 1989).

Já na década de 60, Holland e outros começaram a estudar os chamados sistemas adaptativos, que foram modelados como sistemas de aprendizagem de máquina. Tais modelos conhecidos como algoritmos genéticos (GA na figura 2) implementavam populações de indivíduos contendo um genótipo, formado por cromossomos (que neste modelo eram representados por seqüências de bits), e aos quais se aplicavam os operadores acima citados: seleção, recombinação e mutação. Ainda que Holland tenha proposto um quarto operador (a inversão), este não chegou a ser largamente usado (Davis, 1991).

Uma das primeiras aplicações propostas para os algoritmos genéticos (cuja primazia no uso do termo cabe a Bagley na sua dissertação em 1967), seguindo o caminho pesquisado por Holland, foram os sistemas classificadores (identificados na figura 2 pela folha contendo SCS). Tais aparatos são sistemas de produção, e na verdade usam os algoritmos genéticos em uma parte do algoritmo global. A despeito do interesse que levantaram na época, os sistemas classificadores permanecem como um campo de estudo

ainda inexplorado. Nas palavras de Heitkoetter citando Goldberg, "sistemas classificadores são um pântano. Um maravilhoso e inventivo pântano, mas ainda assim, um pântano (Heitkoetter, 1999).

Outro ramo descendente dos algoritmos genéticos é o da programação genética. Aqui, os indivíduos da população não são seqüências de bits, mas sim programas de computador, armazenados na forma de árvores sintáticas. Tais programas é que são os candidatos à solução do problema proposto. Programação genética não usa o operador mutação e a recombinação se dá pela troca de sub-árvores entre dois indivíduos candidatos à solução.

A seguir tem-se a programação evolutiva, na qual se busca predizer o comportamento de máquinas de estado finitas. Apenas dois operadores foram originalmente usados: a seleção e a mutação. As idéias datam de 1966, e não foram muito consideradas na comunidade na computação evolutiva por rejeitar o papel fundamental da recombinação, embora esta abordagem esteja sendo reavaliada.

Finalmente, completa a árvore a estratégia evolutiva, proposta em meados dos anos 60, na Alemanha. A ênfase aqui é na auto-adaptação. O papel da recombinação é aceito, mas como operador secundário. Segundo Goldberg, "a estratégia evolutiva teve devotos em certos círculos científicos e de engenheiros, particularmente na Alemanha (Goldberg, 1989).

Em 1990, a comunidade da computação evolutiva reuniu esforços para a realização do primeiro evento envolvendo todas as iniciativas. Tratou-se do Workshop on Parallel Problem Solving from Nature, que ocorreu em Dortmund. Este congresso foi sucessivamente realizado, e diversos outros surgiram. Uma medida do interesse crescente deste ramo de estudo e do seu rápido desenvolvimento pode ser observado na newsletter eletrônica Genetic Algorithms Digest, de 17 de fevereiro de 99, (Volume 13 : Issue 4), apresenta nada menos que 32 chamadas de congressos ou eventos científicos diretamente relacionados ao tema.

6.7.2 Evoluindo programas (e não dados)

Nesta parte da CE, ao invés de evoluir dados (através dos indivíduos que representam soluções), evoluem-se programas. Utiliza-se linguagens do paradigma funcional, e aqui serão mostrados exemplos baseados em LISP. Os programas serão expressos na forma de árvores com nodos devidamente rotulados. Nodos internos são funções, predicados ou ações que exigem um ou mais argumentos. Nodos folha são constantes, ações ou funções que não exigem argumentos. Veja-se na figura 11.17 como um programa para computar a expressão $3 + (5 \times 4)/7$ é expresso como árvore.

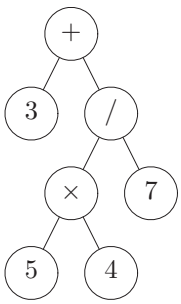


Figura 6.7: Um exemplo de árvore sintática

Neste caso, os nodos folha são as constantes 3, 4, 5 e 7 e os nodos internos são as

 ©88-09, Pedro Kantek

funções +, × e /.

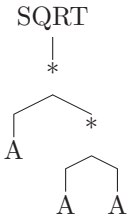
6.7.3 Um exemplo pedagógico

Devido a Mitchell (1996) e citado por Luger (2002). A terceira lei de Kepler do movimento dos planetas descreve a relação entre o período orbital **P** de um planeta e sua distância orbital **A** do sol. A função da lei, usando uma constante **c** é

$$P^2 = c \times A^3$$

.

Se assumirmos que P é dado em anos terrestres e A em unidades de distância Terra-Sol (cerca de 150.000.000 Km), então $c = 1$. Usando notação LISP, a expressão-s desta lei é `P = sqrt(* A ( * A A))`. O programa que se pretende evoluir é

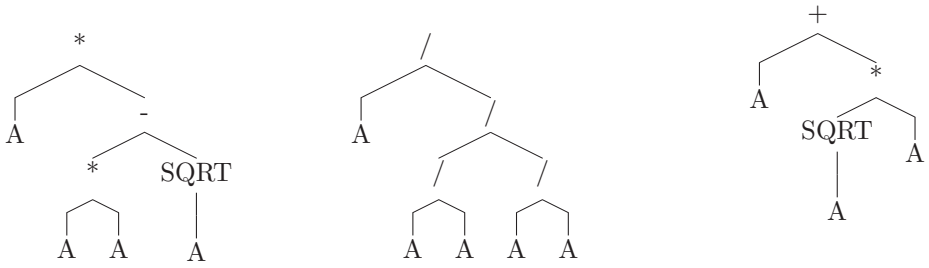


A seleção do conjunto de símbolos terminais neste caso é simples: um único valor real para **A**. O conjunto de funções poderia ser igualmente simples: { +, -, *, /, sq, sqrt}, onde sq é o quadrado e sqrt é a raiz quadrada.

A seguir, cria-se uma população aleatória de programas que poderia ser

```
(* A (- (* A A) (sqr A)))
(/ A (/ (/ A A) (/ A A)))
(+ A (* (sqrt A) A))
```

Esta população inicial tem um limite *a priori* de tamanho e de profundidade, dado o conhecimento de domínio do problema. Eis as 3 árvores dos indivíduos



A seguir, obtem-se um conjunto de testes para a população de programas. Suponhamos um conjunto de dados conhecidos que deseja-se sejam explicados pelo programa evoluído. A tabela 6.7.3 obtida de Urey e publicada em 1952, informa:

Deve-se definir uma medida de aptidão para os indivíduos. Vale a acriatividade aqui. Vai-se usar a seguinte definição: **aptidão = número de saídas que estiverem afastadas em até 20% dos valores de saída corretos**

Com isso vamos valorar os 3 indivíduos da população original

```
(* A (- (* A A) (sqr A)))      aptidão: 1
(/ A (/ (/ A A) (/ A A)))      aptidão: 3
(+ A (* (sqrt A) A))           aptidão: 0
```

Planeta	A	P
Vênus	0,72	0,61
Terra	1,0	1,0
Marte	1,52	1,87
Júpiter	5,2	11,9
Saturno	9,53	29,4
Urano	19,1	83,5

Fica como lição de casa, estruturar o ambiente da programação genética, gerar mais indivíduos da população, construir os operadores de recombinação e mutação, gerar mais gerações, estabelecer critérios de fim, etc etc.

6.7.4 Evoluindo um robot

Vejamos como a PG pode ser usada para evoluir a programação de um robot cujo objetivo é percorrer as paredes de uma grande sala que pode conter objetos grandes (fixos) no seu interior. Sempre existe espaço entre os objetos e as paredes maior do que uma célula. A regra que ele vai usar é caminhar até encontrar uma parede ou um objeto e então seguir pela parede ou objeto para sempre. O robot é capaz de saber se as 8 células adjacentes a ele estão livres ou ocupadas e além disso deve poder executar certas funções primitivas.

Suponhamos que um mundo para este robot seja o mostrado na figura 6.8.

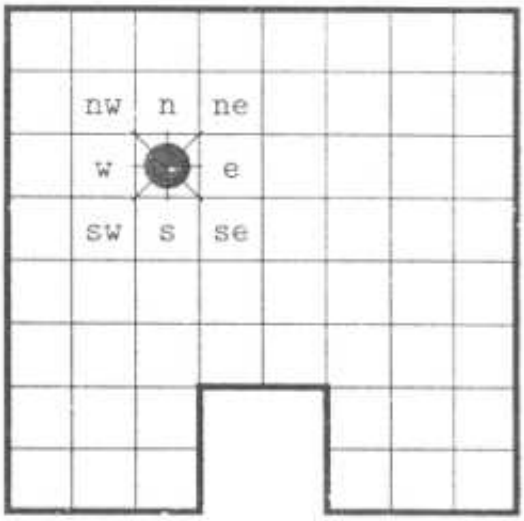


Figura 6.8: Um mundo possível para o robot

Vamos escrever um programa que receba as entradas do robot (8 entradas binárias (0 ou 1) indicando o estado dos 8 vizinhos do robot, e compute qual a ação que ele deve tomar. O robot será controlado pela execução repetida do programa.

As funções primitivas a serem usadas pelo programa incluem as 4 funções booleanadas, tal como definidas no LISP

1. `AND(x,y) = 0` se x=0, senão y

 ©88-09, Pedro Kantek

2. `OR(x,y) = 1` se x=1, senão y
3. `NOT(x) = 0` se x=1, senão 1
4. `IF(x,y,z) = y` se x=1, senão z

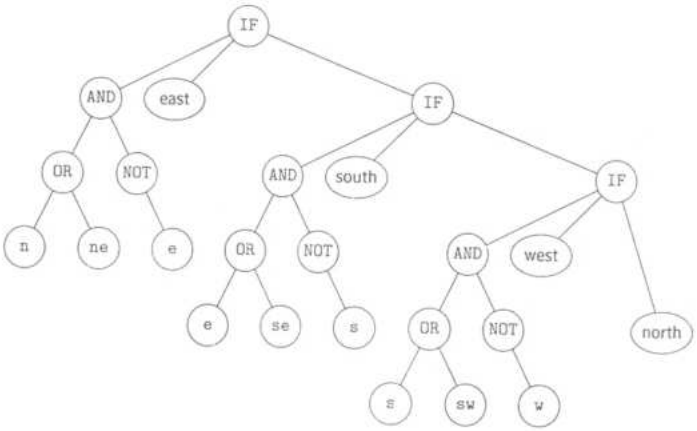
As ações são as seguintes:

1. `NORTE` move o robot uma célula para cima
2. `ESTE` move o robot uma célula para a direita
3. `SUL` move o robot uma célula para baixo
4. `OESTE` move o robot uma célula para a esquerda

As funções não devolvem valor, apenas executam a ordem e interrompem o programa. Se a ordem não puder ser atendida (por causa da parede) o programa termina. Ao invés de usar os 8 indicadores $s_1, s_2, ..., s_8$ vamos usar os mnemônicos n, ne, e, se, s, sw, w e nw. Estes indicadores serão 0 quando a célula vizinha estiver livre para ser ocupada pelo robot e 1 quando não.

Dentro da Programação Genética devemos garantir que todas as expressões e sub-expressões usadas no programa tenham valor para todos os argumentos possíveis, a menos para as expressões que terminam o programa. Por exemplo, ao usar a expressão *Divide* (x, y) para representar o valor de x dividido por y, precisamos prever o valor final (por exemplo, 0) quando y for igual a 0. Assim, garante-se a integridade da árvore formada pelo programa executável.

Antes de evoluir o programa, vejamos um exemplo de árvore e programa na figura 6.9. A execução repetida dele faz com que o robot vá para o norte até bater uma parece e daí siga por ela na direção horária (dos ponteiros do relógio)




```
(IF (AND (OR (n) (ne)) (NOT (e)))
    (east)
    (IF (AND (OR (e) (se)) (NOT (s)))
        (south)
        (IF (AND (OR (s) (sw)) (NOT (w)))
            (west)
            (north))))
```

Figura 6.9: Árvore e programa equivalente

6.7.5 O processo

No começo tem-se uma população de programas randômicos extraídos de um universo de funções, constantes e sensores que se julga serem releventes para o programa em questão. Esta população constitui a geração 0. O tamanho desta população na geração 0 é um dos parâmetros do método. Neste exemplo, os programas randômicos serão construídos a partir das primitivas `AND`, `OR`, `NOT` e `IF`, além dos senores n, ne, e, se, s, sw, w e nw, das ações NORTE, LESTE, SUL e OESTE, além das constantes 0 e 1. Os programas produzidos em cada geração são avaliados e uma nova geração é produzida até que um programa produzido tenha uma performance aceitável.

O programa é avaliado durante sua execução vendo-se como ele desempenha a tarefa para a qual foi designado. Existem 32 células na parede, assim um programa que se saia muito mal terá contagem = 0 e um programa perfeito terá contagem = 32. A seguir o robot é inicializado 10 vezes, em 10 posições aleatórias do tabuleiro e a soma de células limite visitadas é o fitness do programa (podendo variar entre 0 e 320).

A nova geração

A próxima geração é construída a partir da geração atual, com os seguintes critérios:

- 10% da geração atual é copiada para a geração seguinte. Os indivíduos são escolhidos para serem copiados desde que vençam o seguinte *torneio*: sete programas são aleatoriamente escolhidos na população (e posteriormente devolvidos). Destes sete, aquele que tiver o melhor *fitness* é escolhido para ser copiado. O tamanho do torneio (7, aqui) e a percentagem de cópias (10% aqui) são ambos parâmetros do método.

- 90% da nova população é constituída por *descendentes* da população atual. Um descendente é gerado após um "casamento" entre dois indivíduos, chamados ancestrais. Ambos os ancestrais são escolhidos através de um torneio (igual ao acima descrito). Agora os ancestrais fazem a *recombinação*, também chamada *crossover*. Um nodo, randomicamente escolhido do ancestral 1 é trocado por um nodo também randomicamente escolhido do ancestral 2. E vice versa, pelo que o resultado da recombinação é a produção de 2 novos indivíduos. O funcionamento de ambos está garantido pela regra de que todas as funções usadas no programa sejam executáveis para todos os valores dos seus argumentos, ou ainda, de que operações de crossover somente estejam definidos para nodos de mesmo grau. Mostra-se na figura 6.10 a seguir, uma operação de crossover: Os descendentes podem ser (ou não) melhores (isto é, com *fitness* maior) do que seus pais. A explicação para este operador é que ele potencialmente pode juntar um bom esqueleto (main program) a uma boa subexpressão.

- Algumas vezes um terceiro operador, chamado *mutação* é usado para construir indivíduos para a nova geração. Quando usado ele atua pouco (talvez em 1% dos indivíduos, valores maiores degeneram rapidamente a população). Este operador escolhe um ancestral (usando o torneio), e um nodo aleatoriamente escolhido deste indivíduo é excluido e em seu lugar é gerado um novo nodo, de maneira similar a como foram gerados os indivíduos na geração 0.

 ©88-09, Pedro Kantek

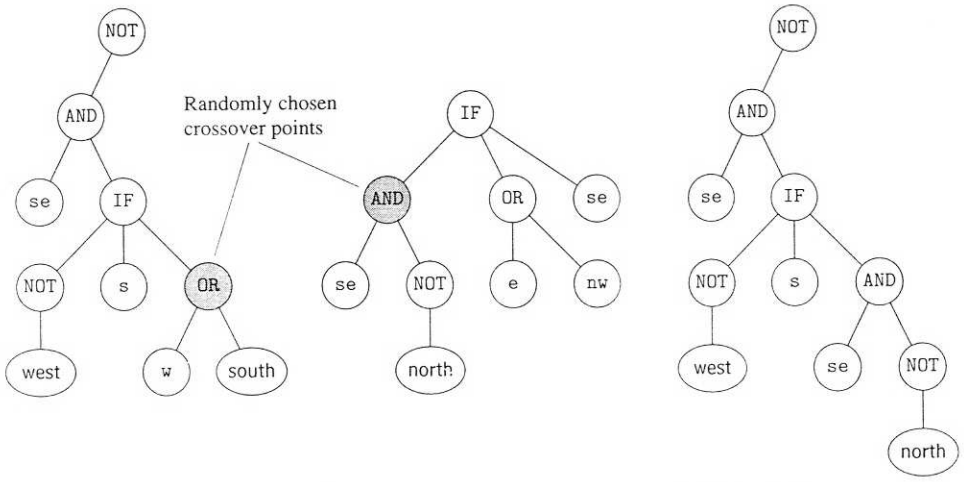


Figura 6.10: Operação de crossover

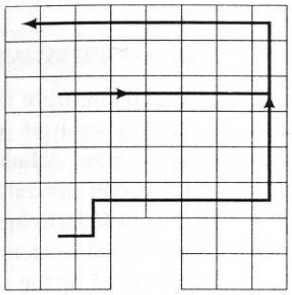
6.7.6 Um caso real

Evoluindo um caso real, com uma população inicial de 5000 indivíduos e usando as técnicas acima descritas, estamos procurando um bom programa de robot para seguir as paredes. Muitos programas (randômicos) da geração 0 não fazem nada. Outros andam um ou dois quadrinhos. Esta geração 0 pode ser considerada como uma busca generalizada sobre o espaço de soluções de programas de computador usando ingredientes escolhidos para esta classe de problemas. O melhor programa da geração 0 e duas de suas rodadas são mostrados na figura 6.11 a seguir Ele tem um *fitness* de 92. Como é comum em PG, o programa é difícil de ler e tem muitas redundâncias. Parte desta dificuldade poderia ser removida por um pós-processador de tradução. Começando em qualquer célula, este programa se move para leste até encontrar a parede. Então ele move norte até poder leste novamente ou então ele se move para oeste até ficar preso na célula superior esquerda.

O melhor programa da geração 2 tem *fitness* de 117. O programa e sua performance em dois casos típicos estão mostrados na figura 6.12 a seguir O programa é menor do que na geração 0, mas ele ainda fica preso na célula inferior direita como mostrado na figura 6.13 a seguir

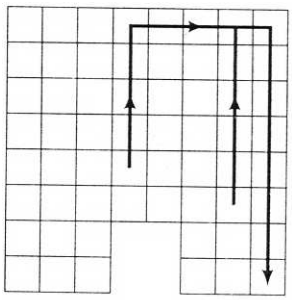
Finalmente, na geração 10, gerou-se um programa que percorre as paredes corretamente. Este programa e dois dos seus caminhos a partir de pontos iniciais diferentes estão mostrados na figura 6.7.6 a seguir

O programa segue a parede no sentido horário e sempre começa se movendo para o sul. Na figura 6.7.6 a seguir, vê-se o desempenho do melhor indivíduo em cada geração. Note o progressivo (ainda que pequeno) aumento de geração para geração.



```
(AND (NOT (NOT (IF (IF (NOT (nw))
                       (IF (e)(north) (east))
                       (IF (west)(0) (south)))
                   (OR (IF (nw)(ne)(w))
                       (NOT (sw)))
                   (NOT (NOT (north))))))
     (IF (OR (NOT (AND (IF (sw)(north)(ne))
                       (AND (south)(1))))
             (OR (OR (NOT (s))
                     (OR (e)(e)))
                 (AND (IF (west)(ne)(se))
                      (IF (1) (e)(e)))))
         (OR (NOT (AND (NOT (ne))(IF (east)(s)(n))))
             (OR (NOT (IF (nw)(east)(s)))
                 (AND (IF (w)(sw)(1))
                      (OR (sw)(nw)))))
         (OR (NOT (IF (OR (n)(w))
                      (OR (0)(se))
                      (OR (1)(east))))
             (OR (AND (OR (1)(ne))
                      (AND (nw)(east)))
                 (IF (NOT (west))
                     (AND (west)(east))
                     (IF (1)(north)(w)))))))
```

Figura 6.11: Melhor programa da geração 0 e duas rodadas

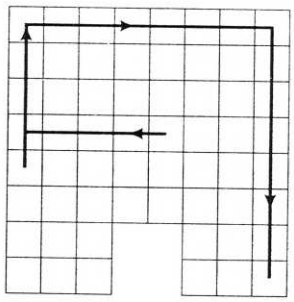


```
(NOT (AND (IF (ne)
              (IF (se)(south)(east))
              (north))
          (NOT (NOT (e)))))
```

Figura 6.12: Melhor programa da geração 2

.

 ©88-09, Pedro Kantek



```
(IF (AND (NOT (ne))
         (IF (e)(s)(nw)))
    (OR (IF (1)(e)(south))
        (IF (north)(east)(nw)))
    (IF (OR (AND (0)(north))
            (AND (e)(IF (e)
                        (IF (se)(south)(east))
                        (north))))
        (AND (e)
             (NOT (IF (s)(sw)(e))))
        (OR (OR (AND (nw)(east))
                (west))
            (nw))))
```

Figura 6.13: Embora melhor, ainda fica preso

6.7.7 Implementação

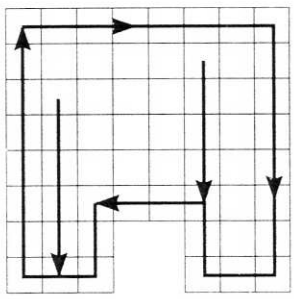
Para implementar este engenho evolutivo, tomaram-se as seguintes definições operacionais.

1. As operações envolvidas são AND (com valência 2), OR (2), NOT (1) e IF (3). Os sensores são 8: *n*, *ne*, *e*, *se*, *s*, *sw*, *w* e *nw*. As ordens são *north*, *east*, *south* e *west*. As constantes são 0 e 1.

2. a definição de AND(x,y) é
   se x=0 **então**
   AND(x,y) vale 0
   senão
   AND(x,y) deve ser substituída por y
   fimse

3. a definição de OR(x,y) é
   se x=1 **então**
   OR(x,y) vale 1
   senão
   OR(x,y) deve ser substituída por y
   fimse

4. a definição de NOT(x) é
   se x=1 **então**
   NOT(x) vale 0
   senão



```
(IF (IF (IF (se)(0)(ne))
        (OR (se)(east))
        (IF (OR (AND (e)(0))
                (sw))
            (OR (sw)(0))
            (AND (NOT (NOT (AND (s)(se))))
                 (se))))
    (IF (w)
        (OR (north)
            (NOT (NOT (s))))
        (west))
    (NOT (NOT (NOT (AND (IF (NOT (south))
                            (se)
                            (w))
                        (NOT (n)))))))
```

Figura 6.14: Melhor na geração 10

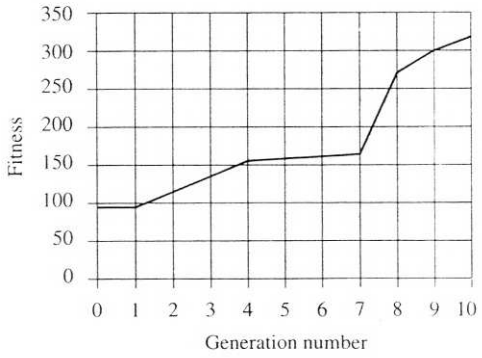


Figura 6.15: Aumento do fitness com o passar das gerações

NOT(x) vale 1
fimse

5. a definição de IF(x,y,z) é
   se x=1 **então**
   IF(x,y,z) deve ser substituída por y
   senão
   IF(x,y,z) deve ser substituída por z

 ©88-09, Pedro Kantek

fimse

6. A avaliação de um programa indivíduo começa com a localização aleatória do robot em (x_i, y_i). O programa deve ser repetido enquanto houver movimento do robot. Quando a aplicação do programa não resultar em movimento, (isto é, $x_{n+1} = x_n$ e $y_{n+1} = y_n$, todo o ciclo deve ser encerrado.

7. Quando uma ordem é usada como entrada, por exemplo em `(OR (e) (west))`, a ordem (neste caso *west*) é executada, e não sobe nenhum resultado na árvore, pelo que o programa pára.

8. Quando um sensor é usado em uma saída (por exemplo em `IF(0 (west) (e))`) todo o IF vale o sensor (neste caso `e`) que é levado para cima na árvore e segue o baile!

9. o programa é avaliado pela ordem (da raiz para baixo), seguindo a avaliação convencional até que o robot se movimente ou o programa termine. Se o robot se movimenta, esta encarnação do programa é encerrada e volta a repetição, agora com uma nova localização do robot. Se o programa termina sem movimento, acabou a avaliação deste indivíduo.

10. O laberinto é sempre 8×8 e é fornecido em uma matriz contendo 0´s e 1´s, onde 1 significa caminho livre e 0 significa obstáculo. Sempre há pelo menos dois espaços contendo 1 entre paredes paralelas (um para ir e outro para voltar). Por exemplo

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1
1	1	1	0	1	1	1	1
1	1	1	0	1	1	1	1
1	1	1	0	1	1	1	1
1	1	1	0	1	1	1	1

11. O workspace se chama VIVO855.

6.8 Orientações para uso do GALOPPS

1. Deve-se buscar o GALOPPS em isl.msu.edu/GA, ou em qualquer outro local encontrado pelo google.

2. Deve-se buscar informações sobre o funcionamento de um algoritmo genético (Goldberg, capitulos 1 a 4).

3. Achar um compilar ANSI C (Borland 3.2 ou DJGPP funcionam bem). Copiar todos os módulos do pacote GALOPPS.

4. Encaminhar a solução do problema que se quer resolver (usualmente esta é a parte mais difícil).

5. Projetar o cromossomo, os operadores e os parametros da rodada.

6. Deve-se criar um arquivo ASCII de parâmetros, com o formato a seguir. Se ele não for fornecido, fará todas essas perguntas. Não é uma boa idéia.

numberofruns	1	Indica o número de rodadas. Colocar sempre 1
quiet	3	Indica o nível de geração de informações. 3 é o menor. Como nó vamos controlar a saida, deixar 3. 0 é a maior.
ckptfreq	20	Intervalo de gerações em que o pacote deve gravar um scheckpoint Sugiro colocar 20
checkptfileprefix	parte1	Nome dos arquivos onde serão gravadas as informações de check point. Até 6 caracteres, não começando por "z".
restartfileprefix		Se fornecido, o GALOPPS pega os dados deste arquivo e vai en frente. Para uma rodada normal, deixar em branco. Neste casc os parâmetros a seguir é que valem.
permproblem	n	Indica se o problema é de permutação. Responder "n".

alpha_size	16384	Indica (de uma maneira meio "diagonal") o tamanho do alelo (campo). Assim, se alpha_size=2 tem-se um campo binário de 1 bit. Se alpha_size=4, são 2 bits. O valor máximo (eu acho) é 16384 que corresponde a 14 bits por campo.
numfields	70	Número de alelos (campos) no cromossomo. Se alpha_size tivesse sido 2, este seria o tamanho do cromossomo.
maxgen	150	Número máximo de gerações
popsize	100	Tamanho da população em indivíduos. A literatura recomenda entre 50 e 200
printstrings	n	Impressão ou não dos indivíduos a cada geração (Se pedir "s", arrume bastante área em disco. Sugiro deixar "n"
pcross	.85	Probabilidade de um casal sofrer crossover. A literatura recomenda entre 0.7 e 0.95
pmutation	.03	Probabilidade de um bit sofrer mutação. A literatura recomenda entre 0.001 e 0.05
pinversion	0.	Probabilidade de inversão. A literatura recomenda que não haja inversão, ou seja, p=0

autocontrol_selection	n	Métodos mais sofisticados de seleção. Deixe "n"
beta	3	Parâmetro de seleção. Não usado se autocontrol...=n. Entre 0 e 3. Deixe 3.
scaling_window	-1	Janelamento. -1 significa não
sigma_trunc	0.	Truncamento de sigma. Deixar 0
scalemult	0.0	Fator de escala. Deixar 0
crowding_factor	0	Fator de crescimento. Deixar 0
conv_sigma_coeff	0.	Coeficiente de conversão. Deixar 0
convinterval	0	Intervalo de conversão. Deixar 0
tourneysize	15	Tamanho do torneio. Importante se for usado o método de seleção do torneio.
randomseed	.555	Muito importante. É a semente aleatório. Deve ser trocada a cada execução. Um real entre 0 e 1.

7. Escrever uma função C de nome "objfunc()"

```
void objfunc(critter)
struct individual *critter;
{
    neval++;
    local_cycle_neval++;
    critter->neval = neval;
    critter->init_fitness = avalia(critter->chrom);
    return;
}
```

 ©88-09, Pedro Kantek

critter é um pointer que aponta para o indivíduo que está sendo avaliado neste instante. Seu "fitness"deve ser colocado no campo critter->init_fitness, ao passo que critter->chrom aponta para o cromossomo.

A função "avalia"recebe um pointer e devolve um float. Nada impede que você imprima (pelo menos no começo, para efeito de acompanhamento e depuração, o valor de cada um dos cromossomos).

8. A função app_after_random_init() pode ser chamada, para fazer algum tipo de inicialização (se não for desejada a inicialização 100% aleatória).

```
void app_after_random_init()
{
  int ik,j;
  struct individual *xaxi;
  int arrak[70];
  for (ik=0;ik<100;ik++) {
      xaxi = &(oldpop[ik]);
      chromtointarray(arrak,xaxi->chrom);
      for (j=0;j<70;j++) {
//------------------------ POPULACAO INICIAL -------------------------
          if (flip(0.9)) { // probabilidade do valor ser igual ao entorno
        arrak[j] = 1489;
          }
      }
      intarraytochrom(arrak,xaxi->chrom);
  }
}
```

9. As funções de conversão entre cromossomo e campos inteiro são as seguintes:

   int ithruj2int (int i, int j, unsigned int *cromo) Trata os bits i até j como sendo um campo binário, converte-o para inteiro e coloca o resultado. O cromossomo tem seus bits numerados de 1 até tamanho_cromossomo. Exemplo:

   ```
   int valor;
   valor = ithruj2int(1,8,critter->chrom);
   ```

   int getfield (int n, int numero_campo, unsigned int *cromo) Recupera o n-ésimo campo de um cromossomo que tenha numero_campo definido. Este valor deve coincidir com aquele especificado em "numfields"dentro do arquivo de parâmetros. O primeiro campo é o campo 1. Exemplo:

   ```
   int valor;
   valor = getfield (8,32,critter->chrom);
   // este cromossomo tem 32 campos
   ```

   chromtointarray (int *intarray unsigned int *cromo) Recupera o cromossomo inteiro. Exemplo

   ```
   int valor[32];
   valor = chromtointarray (valor, critter->chrom)
   ```

10. Incluir um dos arquivos de seleção

srsselect.c	Seleção estocástica remanescente
rselect.c	amostragem estocástica com restituição (roda da roleta convencional)
rnkselect.c	Neste método os indivíduos são avaliados, colocados em ordem e a abertura da roda da roleta tem a ver com a ordem de cada um e não com o seu fitness
suselect.c	amostragem remanescente estocástica com substituição. Segundo a literatura é melhor e mais rápido que a roda da roleta convencional.
tselect.c	Seleção por torneio. O tamanho do torneio é informado nos parâmetros.

11. Incluir o arquivo de mutação (bitmutat.c)

12. Incluir um dos arquivos de crossover

oneptx.c	Crossover de 1 ponto
twoptx.c	Crossover de 2 pontos
unifx.c	Unified crossover

13. Incluir as seguintes funções checkhdr.c, checkrd.c, checkwt.c, ffscanf.c, filestat.c, generate.c, memory.c, random.c, report.c, statisti.c, utility.c, user_in.c, startup.c, mainone.c, além da função objetivo.

14. E os seguintes arquivos header sga.h, external.h, sgafunc.h, sgapure.h

15. Após cada geração, o GALOPPS vai chamar a função app_quiet_report, desde que o programa esteja sendo chamado com quiet=3.

16. Gerar um executável, acender uma vela e chamar o executável com as seguintes opções:

```
-i <arquivo-entrada>Arquivo contendo os parâmetros
-o <arquivo-saida>Arquivo de saída
-c <prefixo dos arquivos de chesckpoint>
-e <prefixo entrada de checkpoint>
-m <arquivo master>Usado apenas em manypops
-p quiet=<nível de saída: 0, 1, 2 ou 3>
-p npops=<número de populações>  Usado apenas em manypops
...
-p pcross=<probabilidade de crossover>
-p pmutation=<probailidade de mutação>
...
-p popsize=<tamanho da população>

Os valores aqui fornecidos substituem aqueles
      existentes no arquivo de parâmetros.
```

6.9 TUDEL

Embora existam pacotes de programação completos e testados para usar e implementar algoritmos genéticos – pelo menos em sua forma canônica, sem hibridização – desenvolveu-se um ambiente no qual eles pudessem ser implementados. A criação deste ambiente, não sugere que tais pacotes não devam ser utilizados. Pelo contrário. Após

 ©88-09, Pedro Kantek

o desenvolvimento deste ambiente, adquiriram-se muito maiores condições de usar tais produtos, até por entender como as coisas de fato ocorrem lá dentro, nas entranhas do algoritmo.

As premissas do desenvolvimento deste ambiente foram obter:

- Um produto real, com habilidades diante de um caso prático qualquer, e não apenas um demonstrador de tutoriais pré-preparados
- Facilidade de modificar a função objetivo
- Um razoável número de operadores genéticos, a fim de permitir testes de desempenho variando-se os operadores
- Facilidade de criar novos operadores, e de modificar o funcionamento do pacote
- Rapidez de obtenção do produto, ainda que às custas do desempenho
- Implementação do conceito de match (casamento) separado do conceito de seleção, para permitir um tratamento qualquer intermediário entre as duas fases
- A possibilidade de uso em modo batch (não conversacional) para permitir que ele rode noite e madrugada adentro sem supervisão direta
- Tamanhos (de população, de indivíduo, de relatório etc) limitado apenas pelos recursos de hardware e software básico

Diante dessas premissas nasceu o tudel. Este é o nome dado ao referido ambiente e é um acrônimo formado pelos nomes de dois pensadores que estabeleceram as bases de cada uma das ciências. Turing (o TU do nome) estabeleceu as bases da computação, e o que é importante, ...antes de existir um computador. A descrição da máquina universal de Turing [por exemplo, em Pen89] é uma das mais concisas, poderosas, formais, elegantes e não menos importante, belas criações intelectuais humanas. A outra parte do nome (o DEL) vem de Mendel, a pessoa que silenciosamente, por longos anos trabalhando com cruzamento de plantas estabeleceu as bases do que hoje é conhecido como genética. Mendel leu seu trabalhos em 1866, mas sua importância só foi reconhecida no século XX. A esses dois benfeitores da humanidade, o primeiro pela morte trágica e o segundo por não chegar a ver a importância de seu trabalho estabelecida, homenageia-se com este nome.

O tudel foi desenvolvido em APL PLUS PC versão 6.4, posteriormente portado para APL2C, versão 5.0.3 (disponível gratuitamente em www.apl2c.com). A superioridade da linguagem APL para o desenvolvimento deste tipo de aplicativo (exploratório, rapidamente produzido, facilmente alterável) é inegável, embora a linguagem costume assustar seus possíveis usuários pela utilização de um alfabeto novo, normalmente não existente em micros e terminais, o Z-code.

6.9.1 Funcionalidades do tudel

O tudel implementa as seguintes facilidades:

- Tamanho de populações e de indivíduos variáveis e determinados pelo usuário
- Implementação do conceito de seleção e de match separados. Os indivíduos que são selecionados vão para um pool de candidatos a "pai"e é ali que os "casamentos"são feitos
- Quatro tipos de seleção de pais. (Obtidos de [Gol89], páginas 121 e seguintes). São eles:

1. Amostragem estocástica com restituição: a aptidão de todos os indivíduos é usada como parâmetro para uma roda da roleta convencional.
2. Amostragem determinística: Este processo é feito em duas etapas. Na primeira a probabilidade de cada indivíduo ser selecionado é obtido de $p_{select} = f_i / \sum f$, ou seja a aptidão daquele indivíduo dividida pela soma de todas as aptidões. O número de cópias deste indivíduo e_i é então obtido a partir de $e_i = n \times p_{select}$, onde n é o tamanho do pool de pais. A parte inteira de e_i gera imediatamente a quantidade de cópias indicada. Na segunda parte do processo, os indivíduos são classificados em ordem descendente pela parte fracionária de e_i. Os candidatos necessários até preencher o pool são obtidos do topo desta lista ordenada.
3. Amostragem remanescente estocástica com substituição: Também é feita em duas etapas. A primeira é igual ao acima descrito. A segunda, usa uma roda da roleta em que a fatia de cada indivíduo é proporcional à parte fracionária de e_i.
4. Amostragem remanescente estocástica sem substituição: Novamente feito em duas etapas. A primeira segue igual. A segunda usa a parte fracionária de e_i como expectância (probabilidade de sucesso) em uma operação de cara-coroa. Se houver sucesso, o elemento vai para o pool. Se não houver, passa-se ao próximo.

- Três tipos de normalização. São eles:
  1. Avaliação pela função objetivo passa a ser igual à aptidão.
  2. Janelamento (windowing), com a possibilidade de se estabelecer o valor mínimo, para se evitar o surgimento de grupos com baixa aptidão dentro da população.
  3. Normalização linear, podendo-se estabelecer o valor máximo da função objetivo (a ser atribuído ao indivíduo que tiver a melhor avaliação) e o decremento para cada um dos sucessores. O tudel impede que haja aptidões negativas, pelo que se este incremento for muito grande, os candidatos a terem aptidões negativas, passam a ter aptidão = 0.
- Elitismo: Desde zero até o tamanho total da população pode ser estabelecido como o número dos melhores indivíduos que passarão à próxima geração. A única restrição é que deve ser um número par. Se não for, o tudel o arredondará para o próximo par.
- Taxas de mutação e de crossover: livremente fornecidas pelo usuário.
- Dois tipos de crossover. São eles:
  1. Crossover canônico, podendo-se fornecer a quantidade de pontos (variando entre zero e o tamanho do indivíduo).
  2. Crossover uniforme, conforme definido em 3.8 com o nome de crossover uniforme.
- Duas possibilidades de população inicial. Na primeira (aleatória) o tudel se encarrega de gerar a população, de maneira completamente aleatória. Na segunda, é fornecida uma matriz de n indivíduos por k bits representando a população inicial.
- A função objetivo é oferecida como parâmetro ao tudel, sendo que cabe a ela tratar a string de bits que será entregue pelo tudel. Cabe a ela também devolver o valor da avaliação. Ela também está escrita em APL, e como esta linguagem é interpretada, a operação de inserção da função objetiva é simples e fácil.

 ©88-09, Pedro Kantek

- No início do processamento o tudel cria um arquivo em disco (no formato texto) contendo os parâmetros da rodada. Feito isso, a cada geração, são incluídos novos dados nesse arquivo. Esses dados dizem respeito a resultados daquela geração. Opcionalmente podem ser listados os "k"melhores indivíduos, acompanhados de seus dados individuais. K também é um parâmetro fornecido.

- Finalmente, a cada execução do tudel um arquivo de 87 caracteres é guardado também em disco. Este arquivo contém todos os parâmetros dessa execução e 3 valores numéricos: o pior, o melhor e a média de avaliação dos indivíduos da última geração calculada. O objetivo deste arquivo é permitir sua carga no ambiente APL para tratamento estatístico desses resultados sem haver a necessidade da redigitação.

6.9.2 Operação do tudel

Carregado o interpretador APL, deve-se chamar o workspace tudel (através do comando)LOAD tudel). Feito isso, pode-se invocar a função principal (e a única que interage com o usuário final) chamada tudel. Os parâmetros de sua chamada são

```
nf tudel ni nb ng ts tn v1 v2 el tm tc cr pc mi tr cl nr
```

A seguir, a descrição de cada parâmetro

Parâmetro	Significado
nf	Nome da função objetivo. Trata-se de uma matriz contendo a representação canônica de uma função APL monádica com resultado explícito. Deve estar escrito entre aspas
tudel	Não é parâmetro e sim o nome da função tudel
ni	Número de indivíduos da população
nb	Número de bits de cada indivíduo
ng	Número de gerações que se quer produzir
ts	Tipo de seleção desejado. Um número inteiro entre 1 e 4 1 = amostragem estocástica com restituição (roda da roleta convencional) 2 = amostragem determinística 3 = amostragem remanescente estocástica com substituição 4 = amostragem remanescente estocástica sem substituição

tn	Tipo de normalização. Um número inteiro entre 1 e 3 1 = aptidão é igual a avaliação 2 = janelamento 3 = normalização linear
v1	Se tn = 1 ou 2 deve ser zero. Se tn = 3, é o valor inicial da normalização linear
v2	Se tn = 1 deve ser zero. Se tn = 2 é o valor mínimo do janelamento. Se tn = 3 é o decremento a ser aplicado à normalização linear. Neste caso é entrado como número positivo
el	Parâmetro de elitismo. Se for 0, não haverá. Se for diferente de zero, indica quantos melhores indivíduos devem passar à próxima geração. Será arredondado para o próximo par, se ímpar
tm	Taxa de mutação. Número real compreendido entre 0 e 1
tc	Taxa de crossover. Número real compreendido entre 0 e 1
cr	Tipo de crossover. Número inteiro entre 1 e 2 1 = crossover convencional 2 = crossover uniforme, conforme descrito em 3.8

pc	Pontos de crossover. Se cr = 1, indica em quantos locais deve ser feito o crossover. Pode variar entre zero e o tamanho do indivíduo. Se cr = 2, este parâmetro não é levado em consideração
mi	Modo de inicialização. Se mi = 0, o tudel inicializa aleatoriamente a geração zero. Se mi = 1, o usuário deve prover uma matriz de nome (delta)COMECO contendo a geração zero
tr	Tipo de relatório desejado. Se tr = 0, o relatório é abreviado, apresentando-se apenas os cl (vide a seguir) melhores cromossomos a cada geração. Se tr = 1, todos os cromossomos são listados
cl	Se tr = 0, indica quantos cromossomos devem ser listados a cada geração nr Número da rodada. Comentário, mas que é importante por ser a primeira coisa que aparece no relatório. Também batiza o arquivo de 87 caracteres anteriormente descrito

Durante a execução, o programa vai mostrando, a cada geração: o número da rodada (parâmetro nr), o número da geração que acabou de ser calculada, o total de gerações pedidas, o valor da média da função objetivo para os indivíduos daquela geração, o melhor valor, e finalmente qual a estimativa do tempo que falta para o encerramento dessa rodada. Essa informação vai sendo atualizada a cada geração para oferecer uma retroalimentação ao usuário.

A saída impressa é guardada em disco, com o nome TChhmmss, onde TC é uma constante, hh é a hora inicial da rodada, mm são os minutos e ss os segundos em que esta execução iniciou. O arquivo de parâmetros com os 3 resultados resumo da rodada é guardado com o nome TRrrrr, onde TR é constante e rrrr é a rodada.

 ©88-09, Pedro Kantek

6.9.3 Exemplo real do tudel

Seja uma rodada assim especificada.

'fun00' tudel 100 44 40 2 3 100 1 2 0.05 0.9 1 3 0 0 4 195

Este exemplo significa que se deseja rodar a função objetivo fun00, que é a função denominada F6, cuja fórmula é

$$M = 0.5 - \frac{(sen\sqrt{(x^2+y^2)^2} - 0.5}{(1.0 + 0.001 \times (x^2+y^2))^2}$$

Esta rodada teve 100 indivíduos de comprimento igual a 44 bits. Deverão ser 100 gerações, com amostragem determinística e normalização linear, com valor inicial igual a 100 e decremento de 1. Depois vem a indicação de elitismo = 2. As taxas são de 5% para mutação e 90% para crossover. O crossover é convencional com 3 pontos. O modo de inicialização é aleatório e o relatório desejado é abreviado com apenas os 4 melhores cromossomos. O número da rodada é 195. Este relatório real se encontra abaixo, e levou cerca de 15 minutos rodando em um microcomputador 386 DX40 com 4 Mbytes de memória, processado em 1994. Em 2003, rodando em um Pentium 4 com 1GHz e 256Mb de memória levou menos de 2 segundos.

```
Universidade Federal de Santa Catarina / Curso de Pos Graduacao em Eng. Eletrica / Disciplina de Algoritmos Geneticos 20/
7/2003

TUDEL v.1.1 - TUring + menDEL = ambiente de desenvolvimento de algoritmos geneticos - Autor P. Kantek 13: 5:30

Listagem resumida (apenas os parametros e os resultados finais) Pag. 1

Num rodada = 195 Individuos = 100 Num. bits = 44 Num. geracs = 40 Tipo sel = 2 Tipo norm = 3 V1 = 100.000 V2 = 1.000

Elitismo = 2 Tx mutacao = 0.0500 Tx Cross = 0.9000 Tipo cross = 1 Pontos = 3 Modo de inicializacao = 0

—— Geracao numero 0 ————————————————————————————————————————

Med fo = 0.51 | Max fo = 0.65 | Med ap = 50.50 | Max ap = 100.00 | DP fo = 0.04 | D

Quantidade de Crossovers = 126 | Quantidade de Mutacoes = 132

Num-Melhores cromossomos ——————————-P1 P2 —-F.Obj—-Aptid Qt Melhores prox. geracao ——————————P1 P2

1) 01100110111000000101001000101101100110111110 0 0 0.65 01100110111000000101001000101101100110111110 0 0

2) 01100001111101010111110111001101011110100010 0 0 0.65 01100001111101010111110111001101011110100010 0 0

3) 01101111100111101100101001111111001100100010 0 0 0.62 01001010100001110010000110110110011110011011 7 10

4) 10011110000000111100010110010011010011101001 0 0 0.61 01111010011010100011011011011110001011010110 7 10

—— Geracao numero 39 ————————————————————————————————————————-

Med fo = 0.90 | Max fo = 0.96 | Med ap = 50.50 | Max ap = 100.00 | DP fo = 0.17 | D

Quantidade de Crossovers = 135 | Quantidade de Mutacoes = 114

Num-Melhores cromossomos ——————————-P1 P2 —-F.Obj—-Aptid Qt Melhores prox. geracao ——————————P1 P2

1) 01111101101001010100101000011110101110100111 0 0 0.96 01111101101001010100101000011110101110100111 0 0

2) 01111101101001010100101000011110101110100111 0 0 0.96 01111101101001010100101000011110101110100111 0 0

3) 01111101101001010100101000011110101110100111 4 46 0.96 01111101101001100010001000011110101110101111 23 24

4) 01111101101001010100101000011110101110100111 1 6 0.96 01111101101000010100101000011110101110111101 23 24
```

6.9.4 Funções APL desenvolvidas

tudel Esta função atua como supervisora do sistema, recebendo os parâmetros. Nas próximas versões do tudel ela supervisionará a consistência dos dados, o que por enquanto não é feito ainda. Cria uma série de variáveis globais cujos nomes começam com o caracter (delta), e que ficam disponíveis para todas as demais funções do ambiente. Cria os arquivos em disco e coloca no primeiro o header, contendo os dados daquela execução. Inicia um ciclo que terminará após a última geração. Dentro do ciclo emite a mensagem de feedback para o usuário. Grava também um arquivo contendo uma única linha: os 16 parâmetros de entrada e 3 informações

numéricas: a melhor, a pior e a média das avaliações dos indivíduos da última geração. A finalidade deste arquivo é permitir sua posterior importação no APL para permitir análises de desempenho sem ter que imprimir relatórios, buscar informações, redigitar valores e conferir tudo isso. Feito isso, ele também imprime (grava) os dados para o relatório completo, e a última coisa do ciclo, é transformar a próxima geração na geração atual antes de voltar ao início do ciclo. Ao final, elimina as variáveis de trabalho e retorna o controle ao usuário (deve-se lembrar que o APL é um ambiente interpretado).

modelo Não realiza nenhum trabalho computacional, servindo para ajudar o usuário a preencher todos os parâmetros da tudel.

geracao0 Calcula a função objetivo para cada um dos indivíduos da geração atual. Ao final deste cálculo, põe os indivíduos em ordem decrescente de avaliação. Gera a média e o melhor valor da avaliação daquela geração. Estabelece para cada indivíduo a sua aptidão baseada no tipo de normalização solicitado (parâmetro t_n sendo 1, 2 ou 3). Esta função se encerra quando a tabela de aptidões para aquela geração está completa.

geracao1 Estabelece o pool de pais, que, por definição, tem o mesmo número de indivíduos que uma geração comum. Os participantes daquela geração vão sendo incluídos no pool de pais, de acordo com o tipo de seleção solicitada (parâmetro ts sendo 1, 2, 3 ou 4). À medida em que cada indivíduo vai sendo selecionado, um contador de seleções, associado à população original da geração vai sendo incrementado. Esta função se encerra quando o pool de pais está completo.

geracao2 Inicia analisando o elitismo. Se houver, os indivíduos do topo da geração atual são levados para a próxima geração. Feito isso, dois pais são aleatoriamente selecionados e passados para a função "gerafilho". Recebendo o resultado (2 filhos) eles são colocados na próxima geração. Essa função se encerra quando a próxima geração estiver completa.

gerafilho recebe dois pais, obtidos do pool de pais, e gera os 2 filhos, aplicando mutação e crossover, de acordo com suas taxas e tipos. Totaliza, para efeitos de estatística, a quantidade de crossover e de mutações que se realizaram. Esta função se encerra quando os dois filhos estiverem gerados. Eles são devolvidos para a função "geracao2".

janelanor implementa o janelamento com valor mínimo opcional. Isto significa:

- achar o menor valor do vetor
- fazer o vetor = vetor menos o menor valor (Ou seja, agora o menor valor é zero)
- depois, os valores que são menores que o parâmetro de janelamento, passam a ter este valor

decrenor implementa a normalização linear. Aqui, o vetor (de valores da função objetivo) é posto em ordem decrescente. O primeiro da fila recebe o primeiro parâmetro. O seguinte vale o anterior menos o segundo parâmetro. E assim sucessivamente até o final. Deve-se cuidar para que não haja

cara função de cara ou coroa, recebe a probabilidade de "cara"(entre 0 e 1) e devolve 1 se der cara e zero se der coroa.

dp calcula o desvio padrão de uma série de números.

 ©88-09, Pedro Kantek

outras funções de listagem, limpeza de variáveis, depuração e toolkit de programação, de interesse apenas aos mantenedores do ambiente.

6.9.5 Estudo de caso

O tudel foi utilizado na análise da função F6, conforme definido em [Dav91]. Esta função reúne condições boas de testar uma implementação de algoritmo genético, uma vez que ela é oscilante, multimodal, com máximo global único, simétrica e assintótica aos dois lados. Seu gráfico pode ser assim visualizado (extraído de [Dav91]).

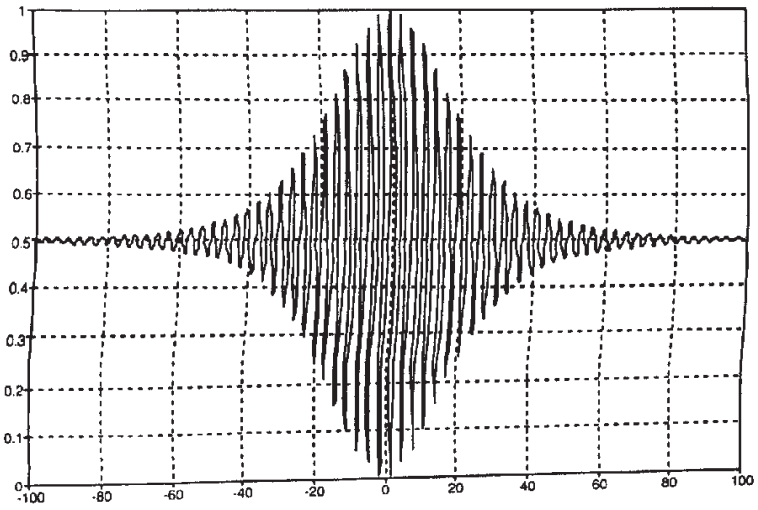


Figura 6.16: Gráfico da figura F6 de Dav91

Eis algumas características do funcionamento da função F6

- Quebra o número binário de 44 bits em 2 números de 22 bits
- Transforma-os em números decimais
- Multiplica-os por 0.00004768372718899898. Esta multiplicação transforma o range dos números de 0 a $2^{22} - 1$, para o intervalo 0 - 200
- Diminui 100 de cada número. O novo intervale está entre -100 e 100
- Aplica estes 2 números na fórmula dada.

O objetivo é otimizar M (achar seu ponto máximo)

Schaffer et alli. escolheram a função F6 para medir a efetividade dos algoritmo genético devido a algumas características interessantes: (1) tem um ponto ótimo, visto na figura 1.3, da pág 10 de [Dav91]. (2) é simétrica, (3) é assintótica à esquerda e à direita. (4) é oscilante (o que leva à breca com tentativas tipo subida da montanha: levam a ótimos locais), (5) tem um só ótimo global.

Entretanto, não há nada de mais especial com F6. Ela pode ser substituida por qualquer outra função.

Ele tem máximo em 1, com valor de x e y igual a 0,5. Logo em ambos os lados, a função decai rapidamente a zero, e depois há uma série de máximos locais cada vez menores, até as assíntotas nos dois lados.

Como o comportamento da função é de antemão conhecido, o objetivo passa a ser conhecer o funcionamento do algoritmo e fazer análises sobre o seu desempenho. Nesse ponto temos uma dificuldade. Olhando por cima, apresentam-se as seguintes possibilidades de teste (quadro obtido por estimativa e usando apenas o senso comum)

Variável	Valor inicial	Valor final	Intervalo	Número de estados possíveis
Número de indivíduos	20	150	10	14
Bits por indivíduo	32	64	4	9
Número de gerações	30	60	5	7
Tipos de seleção	—	—	—	4
Tipos de normalização	—	—	—	3
Valor decr. p/ janelamento	1.9	0.1	0.2	10
Elitismo	0	10	2	5
Taxa de mutação	0.01	0.40	0.02	20
Taxa de crossover	0.5	1.0	0.05	10
Tipos de crossover	—	—	—	2
Pontos de crossover	1	8	1	8

Mediante essa análise estimativa, julga-se necessário efetuar $14 \times 9 \times 7 \times 4 \times 3 \times 10 \times 5 \times 20 \times 10 \times 2 \times 8 = 1.693.440.000$ testes, a fim de poder fazer uma análise de correlação e determinar o que é importante e o que é acessório na análise do desempenho do algoritmo. Se cada teste levou em média 15 minutos, estima-se que seriam necessários 48.325 anos para concluir essa bateria de testes.

Trata-se de um universo amostral muito grande. Tanto é que já houve pesquisadores (Weinberg, por exemplo) que utilizaram algoritmos genéticos para estabelecer parâmetros adequados para algoritmos genéticos. Aqui optou-se (um tanto imperialmente, é verdade) pela seguinte estratégia

- Primeiro determinar quais normalizadores e seletores são mais adequados, incluindo elitismo, usando parâmetros de tamanho (de indivíduo e de população) e taxas (de mutação e de crossover) dentro do que a literatura sugere.

- Num segundo momento, e tendo estabelecido a melhor configuração de normalizador e seletor mais adequada PARA ESTE PROBLEMA, escolher taxas de mutação e de crossover de melhor desempenho.

- Sabendo as taxas PARA ESTE PROBLEMA, conduzir testes variando os tamanhos, buscando responder a questão de até quanto tais tamanhos podem ser diminuídos, sem que se perca qualidade na resposta.

- Finalmente, de posse de todos os parâmetros corretos – ou pelo menos satisfatórios – para o problema, ele deverá ser executado um bom número de vezes a fim de diminuir o efeito estocástico das respostas, substituindo populações de indivíduos por populações de médias.

Agindo dessa maneira, crê-se poder reduzir o trabalho de sugerir parâmetros adequados a algo mais razoável, embora sempre haja o risco de, por não efetuar uma varredura completa sobre o espaço amostral, ficar preso em algum máximo local.

 ©88-09, Pedro Kantek

6.9.6 Análise de normalizadores e seletores

Na primeira fase da análise, foram feitos 44 testes completos buscando responder 3 perguntas, e envolvendo os parâmetros a seguir com valores fixos ou próximos de:
população: 100 indivíduos
gerações: 40
tamanho do indivíduo: 44 bits
Taxa de mutação: próxima de 10 %
Taxa de crossover: próxima de 90 %

As perguntas buscadas nesta primeira fase foram:

- Qual o melhor seletor
- Qual o melhor normalizador
- Qual a melhor taxa de elitismo

Para estudar o resultado, buscou-se apoio na literatura. Davis sugere fazer a análise quantitativa usando-se os melhores cromossomos de cada geração. Não pareceu uma boa estratégia e explica-se porque: Quando se usa elitismo, e considerando que em 40 gerações de 100 indivíduos há grande probabilidade de um bom resultado ser gerado completamente ao acaso. Ora, como algoritmos genéticos não são determinísticos, nada garante que em uma próxima rodada estes valores sejam novamente gerados.

Por essa razão optou-se por uma análise mais conservadora dando importância maior à média de avaliação dos indivíduos e não apenas considerando a avaliação do melhor indivíduo.

Nos 44 testes conduzidos, obteve-se o seguinte resultado:

Seleção	Normalização	Média das médias	Média dos melhores
1	1	0.60518	0.92797
1	2	0.67228	0.97316
1	3	0.77070	0.98302
2	1	0.54292	0.98118
2	2	0.59069	0.94696
2	3	0.85464	0.99634
3	1	0.76564	0.98165
3	2	0.82307	0.99028
3	3	0.86516	0.99352
4	1	0.57885	0.89612
4	2	0.62227	0.96171
4	3	0.83913	0.99302

Generalizando tais resultados, obtiveram-se os quadros a seguir:

Seleção	Normalização	Média das médias	Média dos melhores
1	qualquer	0.67474	0.95942
2	qualquer	0.64531	0.97287
3	qualquer	0.81366	0.98802
4	qualquer	0.66562	0.94640

E

Seleção	Normalização	Média das médias	Média dos melhores
qualquer	1	0.62315	0.94673
qualquer	2	0.67708	0.96802
qualquer	3	0.83243	0.99147

Da análise dos dois quadros vistos acima, percebe-se que PARA ESTE PROBLEMA parece ser mais efetiva a dupla de seletor 3 (amostragem remanescente estocástica com

substituição), com o normalizador 3 (normalização linear com valor inicial 100 e decremento igual a 1). Os valores inicial e de decremento para a normalização foram arbitrados.

Para encerrar esta primeira análise, faltou apenas olhar para a questão do elitismo. Foram conduzidos apenas 2 conjuntos de testes. A primeira bateria sem elitismo e a segunda com um elitismo de 2 indivíduos. Os resultados obtidos foram

Parâmetro de elitismo	Média das médias	Média dos melhores
0	0.69247	0.90197
2	0.70259	0.99094

Os resultados parecem ser inconclusivos, já que o melhor dos indivíduos é melhor quando há elitismo, por razões óbvias. Já na coluna média das médias, percebe-se uma inversão nos resultados, ainda que por um valor pequeno (inferior a 1%). Diante dessa constatação, optou-se por conduzir os testes desta parte em diante com elitismo de 2 indivíduos, baseado muito mais na literatura do que nos resultados numéricos aqui obtidos.

6.9.7 Análise de taxas

Nesta segunda parte da análise foram conduzidos 35 testes, analisando-se o desempenho do algoritmo ao serem variadas as taxas de mutação e de crossover. Os resultados foram os seguintes

Taxa de mutação	Média das médias	Média dos melhores
0.05	0.89831	0.98478
0.10	0.84553	0.99028
0.15	0.77519	0.99027
0.20	0.76111	0.99492
0.25	0.69212	0.99070

E

Taxa de crossover	Média das médias	Média dos melhores
0.70	0.79025	0.99130
0.80	0.81375	0.99028
0.90	0.80807	0.98671
1.00	0.78261	0.99137

Como conclusão destes dois quadros, optou-se PARA ESTE PROBLEMA em usar as seguintes taxas de operadores: para mutação o valor de 0,10 já que este valor apresenta a melhor relação entre a média e o melhor. Taxas menores parecem melhorar a média, mas pioram os melhores. Já taxas maiores pioram significativamente a média embora possam melhorar residualmente os melhores.

Já para a taxa de crossover, por idênticas razões utilizou-se a taxa de 0.80. Não foram testados o tipo de crossover uniforme, e a quantidade de pontos de crossover neste ponto dos testes foi fixada em 1 ponto.

6.9.8 Análise de tamanhos

Agora, já com boa parte das decisões tomadas (seletor, normalizador, taxa de mutação e taxa de crossover), verificou-se o impacto das mudanças de população e de número de

 ©88-09, Pedro Kantek

gerações. Obtiveram-se os seguintes resultados

Tamanho de população	Média das médias	Média dos melhores
70	0.83770	0.99523
80	0.81261	0.97384
90	0.87787	0.99997
110	0.83978	0.99028
120	0.83507	0.99028

E

Numero de gerações	Média das médias	Média dos melhores
35	0.75382	0.96277
45	0.87293	0.99628

Como conclusão desta análise observou-se o seguinte: os valores de população acima de 100 não foram suficiente melhores do que os anteriores (sempre obtidos com população = 100 indivíduos) para justificar o aumento crescente em tempo de processamento. Já os valores menores, especificamente o de 90, são bastante tentadores. Entretanto, agir-se-á com cautela, mantendo-se a população nos 100 indivíduos. Isto é feito, para não perder toda a base de comparação anterior, e principalmente para se basear na literatura, que sugere usar o número de 100 indivíduos com bastante freqüência [Dav91] e [Gol89], por exemplo.

Quanto ao número de gerações, 35 forneceu resultados tipicamente inferiores, pelo que é abandonada. E 45 não trouxe nenhuma melhora significativa, pelo que é abandonada também.

Seja como for, mantem-se o padrão até agora praticado de 40 gerações com 100 indivíduos em cada uma.

6.9.9 Repetição de resultados

Finalmente, a ultima parte deste estudo envolveu a execução de uma bateria de 7 testes completos (cada um com 100 indivíduos e 40 gerações), que apresentaram, cada um os seguintes resultados

Teste número	Média dos indivíduos	Melhor indivíduo
1	0.844961	1.000000
2	0.835683	0.990284
3	0.821232	0.990284
4	0.788461	0.990284
5	0.839173	0.990284
6	0.836771	0.999889
7	0.850508	0.990284
Média	0.830969	0.993044
Desvio Padrão	0.192817	0.004360

Neste momento, o que interessa não são os resultados em sí, mas sua média e principalmente o desvio padrão dos mesmos. Percebe-se que o desvio padrão do melhor indivíduo é muito pequeno (cerca de 0,4%), o que efetivamente garante ter o algoritmo convergido para o ponto máximo. Já o desvio padrão da média dos indivíduos é de 0.19 aproximadamente, o que sugere uma baixa dispersão dos resultados em torno da média.

6.10 Algumas aplicações de Algoritmos Genéticos

Muitos biólogos usaram computadores digitais para simular sistemas genéticos (Barricelli, 57; Fraser, 60; Martin, 60) Embora o objetivo fosse estudar fenômenos natu-

rais, o trabalho de Fraser não foi muito diferente do que hoje é AG. Seu trabalho era: $fenotipo = a + q.a|a| + c.a^3$. Ele trabalhava com um string de 15 bits (5 para a, 5 para q e 5 para c). Ele simulou a descendência e estudou a percentagem de indivíduos com fenótipos aceitáveis. Ele só não percebeu que estava trabalhando com funções de otimização. A paternidade deste campo é dada a Holland, que já em 62 estudava sistemas adaptativos ao ambiente. Reconheceu a necessidade de seleção e sugeriu trabalhar com populações (e não com indivíduos, como até então se pregava). Já em 65 pregava a importância do crossover e de outros operadores genéticos. A teoria dos esquemas é de 68-71 (Holland) e o estudo dos k-bandidos armados é de 73-75 (Holland).

6.10.1 Aplicações de interesse histórico

Jogo dos 6 peões (Bagley)

A primeira vez que o termo *algoritmo genético* surgiu foi na dissertação de Bagley (67). Ele estudou o jogo dos 6 peões (jogado em um tabuleiro de xadrez de 9 casas. Existem 3 peões de cada cor, e o objetivo é alcançar o outro lado). Usou reprodução, crossover e mutação, como é hoje. Não usou o resultado da teoria de esquemas (ele desconhecia isto). Introduziu mecanismos de aptidão linear (escalar) para evitar o fenômeno da dominação por superíndivíduo.

Simulação de células biológicas (Rosenberg)

Na mesma época, por volta de 1967. Pesquisou algoritmos genéticos na sua tese de doutoramento. Privilegiou aspectos de simulação e biológicos, pelo que sua contribuição é as vezes esquecida. Simulou uma população de células com uma rigorosa (e simples) bioquimica e uma estrutura genética clássica (1 gen, 1 enzima). A importância do seu trabalho está na semelhança com otimização e busca de raizes. Ele definiu um string de comprimento finito contendo um par de cromossomos (representação diploide). O string tinha 20 genes com um máximo de 16 alelos por gene. Trabalhou com o fenômeno da concentração química como uma propriedade. (Só trabalhou com 1 propriedade e perdeu a chance de ser o primeiro a trabalhar com função multiobjetivo, o que foi feito depois por Schaffer em 84).

Reconhecimento de padrões (Cavicchio)

Em 70, no estudo *Adaptative search using simulated evolution* passou por 2 problemas: reconhecimento de padrões e subrotina de seleção. Ele não atacou o problema diretamente. Usou AG para desenhar detetores para uma máquina de reconhecimento de padrões de arquitetura conhecida. Seu esquema de reconhecimento envolvia imagens de 25 x 25 pixels binários. Um detector é um subconjunto de pixels. Durante o aprendizado as imagens eram mostradas, e uma lista dos estados dos detectores era armazenada junto com o nome da imagem. Na fase de reconhecimento uma imagem era mostrada e um escore simples calculado. Uma lista com os nomes da imagens ordenados era então construída para a imagem desconhecida. Embora o mecanismo seja simples, o esquema só trabalhava bem quando o conjunto de detectores era bem escolhido para um problema em particular (neste caso o reconhecimento de caracteres). Então, o problema se reduziu a achar um bom conjunto de detectores. Achou uma média de 110 detectores por unidade (um desenho particular) com entre 2 e 6 pixels por detector. Cromossomos foram estabelecidos como conjuntos positivos e negativos, por exemplo: +5 +372 +9 -518 -213 -35 -76 +44 +348... significando que no primeiro detector estavam ativos os bits 5, 372 e 9. No segundo 518, 213, 35 e 76... Note-se o alfabeto de alta cardinalidade e o modo não usual de codificação do cromossomo. Se fosse usada uma estrutura

 ©88-09, Pedro Kantek

binária, o cromossomo teria 3581 bits. (um dos maiores problemas já tentados usando algoritmo genético). Usou reprodução e crossover como hoje, só que o corte só podia cair em fronteira de grupo (entre + e - ou - e +). Devido a estrutura genética variável e a alta cardinalidade do alfabeto inventou 3 mutações:

M1 swap de um único pixel dentro de um detector

M2 swap de todos os pixels dentro de um detector

M3 troca de associação de pixels entre detectores adjacentes

Também permitiu inversão, crossover de 2 pontos e duplicação intracromossomica Um mecanismo inovativo foi o esquema de pre-seleção: uma boa descendência é substituída por 1 dos pais para manter a diversidade da população (este era um problema por causa do baixo tamanho de população que ele teve que usar: entre 12 e 20 indivíduos) Usaram-se variáveis globais para para ajustar os parametros centralizadamente. É hora de parar e perguntar: Com que propriedade podem-se usar dados centralizados em qualquer algoritmo adaptado das ciências naturais ? Em uma população biológica, onde estão os cordões, a inteligência que sabe tudo ou a deidade ? No mundo biológico isto é resolvido elegantemente codificando-se os elementos de controle dentro do próprio cromossomo. Se fizermos a mesma coisa nos algoritmos artificiais, de uma só tacada eliminamos a necessidade de controle central e evitamos o dilema da regressão infinita. A despeito desta inconsistência, ele mostrou as vantagens da adaptação de parâmetros comparando com a experiência quando eles permanecem constantes.

algoritmo genético metanivel, simulação de células e Weinberg

Em 70, Weinberg concluiu sua dissertação *Simulação em computador de uma célula viva*. Também seu trabalho é as vezes esquecido pela ênfase em biologia. Ele propôs em detalhe (embora não tenha simulado) um interessante problema de otimização de algoritmo genético. Ele propôs o uso de um algoritmo genético em multicamadas para selecionar um bom conjunto de 15 taxas constantes que controlariam o trabalho com células (simuladas) de escherichia colli. Weinberg também não resistiu ao canto da sereia da auto-adaptação de parâmetros no algoritmo genético. Ele usou um algoritmo genético para adaptar os parâmetros de outro algoritmo genético. O primeiro foi chamado de programa genético não adaptativo, e o de baixo foi chamado adaptativo. No primeiro nível, uma população de 10 strings codificava as probabilidades de seleção, troca cruzada e mutação. Estes valores eram passados para o segundo nível, que gerava 40 indivíduos e com estas taxas, submetia-os à evolução. O desempenho era passado ao primeiro nível para a adaptação neste nivel. Ele estava certo de que a informação centralizada neste esquema era equivalente à interferência de uma autoridade central onipotente. Como biólogo, Weinberg provavelmente estava preocupado com a intervenção de algo parecido com Deus no seu esquema. Apesar de que ele falou muito sobre simulação no seu trabalho, ele não apresentou os resultados de sua simulação do seu sistema na dissertação. A implementação de algoritmo genético metanível teria que esperar pelo trabalho de Mercer (77) e Grefenstette (86).

Hollstien e a função de otimização

O primeiro que aplicou algoritmo genético a um problema puro de otimização matemática (na verdade a um conjunto de 14 problemas) foi a dissertação *adaptação artificial genética em sistemas de controle computadorizados*. O nome não foi muito bom, sugerindo a aplicação de algoritmo genético ao controle retroalimentado digital de alguma

planta industrial. Ainda que ele tenha aludido a esta possibilidade, seu trabalho baseou-se na otimização de funções de duas variáveis ($z = f(x, y)$) usando dominância, crossover, mutação, e a criação de numerosos esquemas baseados em práticas tradicionais de cruzamento de animais e horticultura. Ele investigou 5 diferentes métodos de seleção:

1. Teste da prole: aptidão da descendência controla a subsequente procriação de pais
2. Seleção individual: aptidão de um indivíduo controla o seu uso futuro como pai
3. Seleção familiar: aptidão da família controla o uso de todos os membros da família como pais.
4. Seleção dentro da família: aptidão de indivíduos dentro da família controla a seleção de pais para criação dentro da família
5. Seleção combinada: Dois ou mais dos métodos acima.

Ele considerou 8 métodos de estabelecer o casamento

1. Casamento randômico: todos os casamentos tem igual probabilidade de ocorrer
2. Criação interna: pais aparentados são intencionalmente casados
3. Criação em linha: Um único indivíduo (bem valorado) é cruzado com uma população base e sua descendência subsequente é selecionada como pais
4. Criação externa: Indivíduos com fenótipos muito diferentes são selecionados como pais
5. Auto-fertilização: Indivíduos são cruzados com eles mesmos
6. Propagação clonal: uma réplica exata do indivíduo é produzida
7. Casamento com grupos positivos: indivíduos parecidos são cruzados
8. Casamento com grupos negativos: indivíduos não parecidos são cruzados

Para testar os efeitos de diferentes seleções com esquemas de casamento, Hollstien simulou diferentes combinações das 5 seleções e 8 estratégias sobre 14 funções de 2 variáveis. Os strings tinham 16 bits, onde 2 parâmetros de 8 bits cada eram codificados como ou inteiros sem sinal ou códigos de cinza.

Comparação entre códigos binários e códigos de cinza

Inteiro	Binário	Cinza
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Note-se que nos códigos cinza, inteiros adjacentes diferem apenas por um único bit

 ©88-09, Pedro Kantek

(distância de Hamming de 1). Isto terá vantagem durante a mutação. Independente do código usado (ele propôs também códigos de Hash) os resultados eram mapeados linearmente no intervalo real [0,100]. Ao final dos testes ele concluiu que os esquemas mais robustos eram a criação interna recorrente e o plano de criação cruzada. Identificou os benefícios de usar os códigos cinza, para diminuir a perturbação causada pela mutação. Reconheceu os problemas de usar população pequena (n=16) e sugeriu maiores para o futuro

Frantz e o efeito posicional

Frantz usou grandes populações (n=100) e grandes strings (l=25) Epistasia = supressão da expressão de um gene por outro gene situado num lócus diferente do correspondente ao alelo do gene suprimido. no estudo sobre o efeito de não-linearidades posicionais (epistasias) na otimização de algoritmos genéticos Usou roleta, crossover e mutação. Mas não consegiu demonstrar que posicionamento firme é melhor ou pior que posicionamento frouxo já que usou funções simples GA-decepção: modelos simples não confirmam hipóteses audaciosas...

Gens reais: Bosworth, Foo e Zeigler

A atividade com algoritmos genéticos, estava na crista em 72, pareceu bifurcar-se em 2 estratégias de codificação:

- Minimalista, a partir da teoria dos esquemas de Holland, prefere alfabetos de baixa cardinalidade
- Alfabeto máximo, onde um gen = um parametro, ao invés de usar diversos alelos

Um exemplo deste último é o trabalho dos 3 aí de cima. Seus cromossomos eram compostos entre 4 e 40 parametros reais. Como já se sabia de trabalhos anteriores com alfabetos de alta cardinalidade, um operador de mutação era dificil de obter, e com isso, 5 operadores foram usados

Operadores de mutação sofisticados

Eis alguns: mutação de Fletcher-Reeves (um hill climbing sofisticado), mutação randômica uniforme, mutação aproximação quadrática gaussiana, mutação aproximação cúbica gaussiana, mutação zero.

É uma grande distância da noção de mutação simples exposta até aqui. E também está distante de qualquer precedente biológico razoável. Embora a técnica de alfabeto de alta cardinalidade possa ter importância em algumas funções específicas, ele reduz o paralelismo implícito, e portanto não devemos chamar a isto de algoritmo genético no sentido dado por Holland.

Operador evolucionário: Box

No trabalho de Box (57) sugeriu-se o operador evolucionário. Mais do que um algoritmo era uma técnica de gerenciamento que permitia a trabalhadores não muito espertos realizar um plano de experimentação no entorno de um ponto operacional designado. O objetivo era melhorar (otimizar) uma determinada métrica. Ao longo do ponto inicial controem-se retas ortogonais entre sí (formando um hipercubo). Andar em uma reta implica em alterar uma medida (e uma só) da experiência. Todos os vértices são visitados, e se algum ponto vizinho é melhor, ele passa a ser o novo ponto operacional, e novo hipercubo é criado. (nada mais é do que uma pesquisa por máximos locais). Não é algoritmo genético em seu sentido moderno

Outras técnicas evolucionárias de otimização

Bledsoe (61), Bremermann (62), Friedman (59) apresentaram um ou outro aspecto de algoritmo genético no sentido moderno. Um desenvolvimento independente na Universidade Técnica de Berlim (Rechenberg, 65 e Schwefel, 81) chamado Evolutionstrategie teve seguidores em certos circulos.

Programação evolucionária: Fogel, Owens e Walsh

A rejeição deste trabalho pela comunidade de IA mais do que qualquer outro fator foi responsável pelo ceticismo enfrentado por AG do fim dos 60 até o meio dos 70. Em seu trabalho uma variedade de tarefas previsivas de símbolos sequenciais era executada pesquisando-se através de um espaço de pequenas máquinas de estado finito. Tais máquinas possuem estados, recebem 1's ou 0's e devolvem mensagens (α, β,...) antes de ir para outro (excepcionalmente o mesmo) estado. A máquina foi treinada para predizer ciclos repetitivos de símbolos de saida usando as técnicas da programação evolucionária, que basicamente são os operadores

seleção e mutação: Na forma simples, a seleção escolhe as 2 melhores máquinas para obter descendência e mutação. População maior que n=2 foi considerada, mas a mutação era o único operador que modificava a estrutura. A mutação era uma modificação em relação à máquina pai e podia diferir em um símbolo de saida, um estado, o número de estados, ou o estado inicial. Em resumo era uma técnica de alteração randômica de uma máquina de estados finita com uma seleção do tipo salve-a-melhor.

Otimização de funções: DeJong

Além do livro de Holland em 75, ficou pronta a dissertação *Uma análise do comportamento de uma classe de sistemas genético adaptativos*. Este trabalho permanece como marco até hoje, pela combinação dos esquemas de Holland com computações cuidadosas. Ele entendia AG como ferramenta de otimização e buscava novos domínios: design de estruturas de dados, de algoritmos, de sistemas operacionais. Levantou a importância de experimentação controlada em novos domínios Seu ambiente de teste foi construido sobre 5 problemas de minimização. As funções tinham em comum:

- contínuas / descontínuas
- convexas / não convexas
- unimodais / multimodais
- quadraticas / não quadráticas
- baixa dimensionalidade / alta dimensionalidade
- deterministicas / estocásticas

Funções usadas

Nome	Função	Limites
F_1	$f_1(x_i) = \sum_1^3 x_i^2$	$-5.12 \leq x_i \leq 5.12$
F_2	$f_2(x_i) = 100(x_1^2 - x_2^2)^2 + (1 - x_i)^2$	$-2.048 \leq x_i \leq 2.048$
F_3	$f_3(x_i) = \sum_1^5 int(x_i)$	$-5.12 \leq x_i \leq 5.12$
F_4	$f_4(x_i) = \sum_1^{30} i.x_i^4 + Gauss(0,1)$	$-1.28 \leq x_i \leq 1.28$
F_5	$f_5(x_i) = 0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}$	$-65.536 \leq x_i \leq 65.536$

As funções usadas estão na tabela da pág 103 de Gol89. Para quantificar a efetividade dos diversos algoritmos genéticos, ele fez duas medidas: uma para medir a convergência

 ©88-09, Pedro Kantek

(chamou-a offline) e outra para medir a performance (on-line). Em uma aplicação off-line, muitas avaliações de funções podem ser simuladas, e a melhor alternativa é depois usada. No on-line, não há este luxo, e a função é avaliada durante um experimento real. Ele definiu a performance on-line $x_e(s)$ da estratégia s no ambiente e como $x_e(s) = \frac{1}{T}\sum_1^T f_e(t)$

onde $f_e(t)$ é o valor da função objetivo para o ambiente e na tentativa t. Em palavras, a performance on-line é uma média de todas as valorações da função, incluindo a atual. Também definiu a performance off-line $x_e^*(s)$ da estratégia s no ambiente e, como $x_e^*(s) = \frac{1}{T}\sum T1 f_e * (t)$ onde $f_e^*(t) =$ o melhor $fe(1), fe(2), ...fe(t)$ Em palavras a performance off-line é uma média das melhores performances em um dado tempo particular. Ele começou com uma versão chamada R_1 (reprodutive plan 1) que considerou 3 operadores:

1. seleção de roda da roleta
2. crossover simples (com casamento randômico)
3. mutação simples

Aqui, vale a observação de que seleção e casamento para o Goldberg são coisas diferentes. (Não são para o Lawrence Davies. Aqui, a seleção via roda da roleta significa gerar tantas cópias quanto seja a aptidão do pai. Colocadas todas estas cópias em um pool, o casamento vem depois, pela seleção dos dois pais para a descendência. Lá no Davies, a seleção da roda da roleta já escolhe um pai, e palicando-se de novo a roda da roleta obtem-se o outro pai, e voilá, temos um casal. R_1 não era um plano único, mas uma família deles dependendo de 4 parâmetros

- n= tamanho da população
- pc = probabilidade de crossover
- pm = probabilidade de mutação
- G = gap generacional

G permite o overlaping de populações. Está definido entre 0 e 1, como segue: G = 1 populações não se overlappam $0 < G < 1$ populações se overlappam Quando há overlap nG indivíduos são selecionados para sofrerem ação genética. O resultado (a descendencia) é colocada na população existente pela escolha de nG slots randomicamente (Fenômeno parecido está descrito no Davies com o nome de *steady-state reproduction* – não há a troca total da população, só alguns indivíduos de cada vez).

No seu estudo DeJong considerou a variação de cada parâmetro individualmente e depois em combinações limitadas em F1 (função de 3 variáveis). Suas conclusões Populações grandes alcançam boa performance off-line. Devido a sua inércia tem mau desempenho on-line inicial Para combater a perda de alelos é frequentemente sugerido aumentar a taxa de mutação. DeJong mostrou que isto não é uma panacéia O incremento de pm degrada a performance on-line Sugeriu uma probabilidade de crossover de 0,6 (só 60% dos indivíduos que se casam sofrem crossover) como um compromisso entre performance on-line e offline. Mercer e Grefenstette sugeriram depois 1,0 (todos sofrem crossover) Sugeriu que populações sem overlapp são melhores em muitos experiementos (principalmente onde performance off-line é o desejado. Entretanto, a performance on-line não é severamente degradada usando pequenos valores de G, este fato é importante por exemplo em machine learning. Para melhorar os resultados, DeJong estudou 5 variações de R1:

- R2: modelo elitista

- R3: modelo do valor esperado
- R4: modelo do valor esperado elitista
- R5: modelo do fator aglomerado
- R6: modelo do crossover generalizado

R2: Modelo elitista:

Se o melhor indivíduo da geração t não permaneceu na geração t+1, ele é incluido como o membro N+1.

Conclusão: nas funções monomodais o elitismo melhora as 2 performances. Na função F5 (multimodal), ele piora as 2. Em resumo: melhora a visão local em detrimento da visão global.

R3: Modelo do valor esperado: Diminui os erros estocásticos da seleção via roda da roleta. Lembremos que esperamos um aumento (diminuição) exponencial de um esquema se ele tem média de aptidão maior (menor) do que a população. A cada vez que um indivíduo é selecionado para casamento e crossover, seu contador (de filhos esperados) é diminuido de 0,5. Se ele é selecionado para reprodução sem casamento e crossover, ele tem seu contador diminuido de 1,0.

R4: Modelo do valor esperado elitista Uma combinação dos modelos R2 e R3.

R5: Modelo do fator aglomerado Definiu um fator CF (crowding factor). Aqui quando um indivíduo nasce, um outro é escolhido para morrer. O indivíduo a morrer é escolhido de um subconjunto de CF elementos escolhidos randomicamente sobre a população.

R6: Modelo do crossover generalizado Existe um parâmetro chamado número de crossover points (CP). Quando CP=1, temos um caso de crossover simples. Para valores maiores, o string é considerado circular, sem início nem fim.

Melhoramentos nas técnicas básicas DeJong colocou algoritmos genéticos e suas aplicações em bases firmes. Depois do seu estudo diversas pessoas sugeriram melhoramentos. Examinaremos as questões de seleção, escala e métodos de ranking

Esquemas de seleção alternativos

A dissertação de Brindle (81) examinou 6 esquemas

Amostragem determinística Amostragem determinística é um esquema onde as probabilidades de seleção são calculadas como o usual $p_{select} = f_i / \sum f$. Então o número esperado de indivíduos de cada string ei é calculado como $p_{select}.n$. Cada string tem tantas cópias quanto a parte inteira de e_i, e a população é classificada de acordo com a parte fracionária dos valores de e_i. O resto dos indivíduos necessários para preencher a população é obtido do topo da lista classificada.

Amostragem remanescente estocástica sem substituição Os métodos estocásticos remanescentes (com e sem substituição) começam de maneira idêntica ao primeiro. Depois que chega a hora da parte fracionária de e_i, temos que as partes fracionárias são usadas como probabilidades para sucesso em operação de cara ou coroa. O processo segue até preencher totalmente a população.

Amostragem estocástica sem restituição Amostragem estocástica sem restituição, é o *modelo do valor esperado* de DeJong.

Amostragem remanescente estocástica com substituição Começam de maneira idêntica ao primeiro. Depois que chega a hora da parte fracionária de e_i, temos que as partes fracionárias são usadas como pesos para uma roda da roleta, que é usada para completar a população

 ©88-09, Pedro Kantek

Amostragem estocástica com restituição Amostragem estocástica com substituição é um nome ornamental para a velha seleção de roda da roleta.

Torneio estocástico (ranking de Wetzel)

Sugerido em 83 por Wetzel. Aqui as probabilidades de seleção são calculadas normalmente e sucessivos pares de indivíduos são selecionados via roda da roleta. Após obter um par, o indivíduo com maior aptidão é declarado vencedor, inserido na nova população e o outro é desprezado. O processo continua até que a população esteja completa. As funções usadas por Brindle tinham picos na mesma ordem de magnitude do que o número de pontos da sua codificação (230). Isto resultou no que é chamado de problema de aliasing. Amostragens insuficientes de pontos equidistantes resultam em falsas periodicidades e a inabilidade de discriminar entre diferentes picos através da exploração de importantes similaridades. Como conclusão ficou que a seleção remanescente estocástica sem substituição ficou sendo largamente usada (é aquela que honra as partes inteiras e usa as fracionárias como probabilidade de cara em uma cara-ou-coroa, até preencher a população).

Mecanismos de escala

Sem escala, a tendência é que super-indivíduos dominem a população. Podem ser citados:

Escala linear. Já foi discutida. Usa-se uma função escalar $f' = af + b$, onde a e b são escolhidos para fazer com que o melhor indivíduo tenha aptidão multipla (usualmente dupla) do que a média da população. Só se deve cuidar com valores de f' negativos.

Truncação sigma . Usa informações sobre o desvio padrão, antes da linearização. Com isso afasta-se o perigo de números negativos.

Escala da lei da potência. Gillies sugeriu usar uma lei da potencia para obter f' . Ele sugeriu $f' = fk$, e usou k = 1,005, mas em geral k é dependente do problema.

Procedimentos de ranking As dificuldades destes procedimentos de escala, levaram Baker (85) a considerar um procedimento não paramétrico para a seleção. Neste método a população é classificada pela sua aptidão, e indivíduos escolhidos baseados apenas na sua posição nesta lista.

Aplicações correntes

otimização de dutos Tese de doutorado do Goldberg. Problema de condutos de gas natural. O sistema tinha 10 compressores e 10 tubulações ligadas em série. O problema era minimizar o consumo de potência sujeitando-se a pressões máximas e mínimas, bem como a restrições de razão entre pressões. A variável de controle foi a diferença quadrática de pressão na entrada (sucção) e na saida (descarga) de cada estação ($U_i = Pd_i^2 - Ps_i^2$). U foi mapeado em 4 bits, e o string completo tinha 10 códigos de 4 bits para cada uma das 10 estações. Foram feitos 3 experimentos, com n=50, pc=1.0 e pm = 0.001. Os resultados estão na página 131 (melhor resultado) e 132 (média) do Gol89.

Otimização estrutural via algoritmo genéticos Calcular uma treliça de 10 peças. O objetivo é minimizar o peso da estrutura sujeito a restrições de stress máximas e mínimas em cada peça. As variáveis de controle foram Ai, variando entre 0,1 e 10 polegadas quadradas, e codificadas como séries de strings de 4 bits. O resultado está na página 137, mas percebe-se que após 40 gerações o resultado se aproxima bastante do ótimo (adrede calculado por métodos já existentes).

Registro de imagens médicas com algoritmo genéticos Usado em um sistema de DAS (digital subtraction angiography). Neste exame ve-se (através de raio-X) o interior de

uma artéria antes e depois de colocar um contraste. As duas imagens são digitalizadas (100 x 100) e a subtração das duas devem mostrar o conteúdo da artéria. Só que pequenos movimentos do paciente causam disturbios na diferença. Logo as imagens devem ser alinhadas ou REGISTRADAS antes de calcular a imagem diferença. É aqui que entram os algoritmo genéticos. A imagem pré-injeção é transformada através de um mapeamento bi-linear como mostrado no texto e na figura da página 139. A forma matemática da transformação é fixa, mas os coeficientes são desconhecidos. O algoritmo genético foi usado para minimizar as diferenças entre as imagens antes e depois da injeção. As coordenadas x e y dos cantos de cada imagem foram codificados em substrings de 8 bits e linearmente mapeados entre -8 e +8 pixels de deslocamento. Note-se que para calcular a diferença entre as imagens são necessárias 10.000 transformações. Os autores fizeram a hipótese de que o resultado poderia ser obtido com um número menor de pixels. Para testar isto fizeram experiências sumarizadas na figura da pág 139, e chegaram a conclusão que bastam 10 transformadas e não 10.000.

Dilema do prisioneiro Problema clássico, em que 2 pessoas são presas e mantidas incomunicáveis, devendo tomar uma decisão. Se as duas calarem, ambas ganham uma penalidade leve. Se ambos falarem, a penalidade é média e se um deles falar e o outro calar, o que falou não perde nada, e o que calou leva um baita ferro. Vejamos no quadro

Decisão	Jogador B: Cala	Jogador B: Fala
Jogador A: Cala	Pena A = 3, Pena B = 3	Pena A = 0, Pena B = 10
Jogador A: Fala	Pena A = 10, Pena B = 0	Pena A = 7, Pena B = 7

Neste jogo, 2 prisioneiros devem fazer uma escolha: cooperar com o outro (permanecendo em silêncio sobre um crime) e receber uma sentença de 3 anos, ou tentar obter um perdão dedando o outro. Desafortunadamente, você não sabe o que o outro prisioneiro está fazendo. Se você coopera e ele te deda, você fica com 10 anos, e ele sai livre. Se os 2 dedam, ambos saem com pena de 7 anos. [Rio92]

O problema fica mais interessante quando há vários jogos com o(s) mesmo(s) jogador(es). Uma estratégia simples é chamada TIT-FOR-TAT (olho por olho). Aqui, o jogador coopera no primeiro lance e depois responde com a mesma coisa que o seu adversário jogou. Esta estratégia é poderosa embora simples. Axelrod (85, 87) buscou outra, tanto ou mais poderosa, usando algoritmo genéticos. Ele considerou apenas 3 jogos para trás, e em cada um deles há 4 opções:

Jogador A; Jogador B	Letra Identificadora
Cala; Cala	R
Cala; Fala	S
Fala; Cala	T
Fala; Fala	P

Ele codificou o resultado dos últimos 3 jogos em 3 letras RRR, RDD, TTT ... Cada letra usou 2 bits para ser codificada: R=0, S=1, T=2, P=3. Como são 3 letras, temos números binários entre 0 e 63. Daí, Axelrod, codificou 64 Fala & Cala, onde o Fala&Cala índice - i, corresponde à i-ésima seqüência de 64 resultados dos 3 jogos anteriores. Exemplo, Se a posição 0 de Fala&Cala tem um Fala, isto deve ser entendido como a seqüência RRR → Fala. O começo do jogo é indeterminado. Para contornar este problema, Axelrod acrescentou 6 bits indicando falas ou calas, preenchidos sequencialmente apenas para fazer uma premissa sobre o comportamento dos jogadores antes do jogo começar. Com isso o string ficou com 70 bits (6+64). Para testar o ambiente, ele achou 8 oponentes obtidos dos seus torneios em computador. Juntos, estes 8 oponentes representam 98% do comportamento observado. Em uma população de 20, cada um dos strings jogou contra os 8 oponentes. em jogos de 151 movimentos. O fitness foi medido pelo número médio de pontos contra os 8 jogadores.

De um início randômico, o algoritmo genético descobriu estratégias melhores que a

tit-for-tat.

Capítulo 7

LISP

7.1 Introdução

[1]

> Se você quer ir a París, precisa aprender francês. Se quer ir para a Inteligência Artificial, precisa aprender LISP.

7.1.1 Programação Funcional

Relembrando os paradigmas usuais de programação são:

Imperativo O computador é uma máquina de estados que são alterados através de comandos emitidos pelo programador. É o principal paradigma, por estar fortemente associado à arquitetura de Von Neumann . Seus representantes: FORTRAN, COBOL, BASIC, PL/I, PASCAL, C, CLIPPER.

Orientado a objetos Coleção de objetos cujos estados se alteram pela execução de ações associadas a eles. Representantes (híbridas: C++, Object Pascal; puras: Smaltalk, Java)

Lógico Um programa é uma relação. O resultado é obtido computando-se os valores que satisfazem a uma relação. O representante solitário é o Prolog.

Funcional A computação é obtida pela aplicação de funções (no sentido matemático) a seus argumentos. Exemplos: APL,LISP, Haskell.

Comparando o paradigma funcional com o imperativo, que é o mais conhecido, pode-se afirmar:

- As linguagens funcionais estão baseadas fortemente em funções. Programas são funções. Funções feitas pelo programador depois de prontas são indistinguíveis das funções primitivas.
- O paradigma é a função matemática, com o conceito de uma correspondência biunívoca entre os conjuntos imagem e domínio. (em $y = f(x)$, x é o domínio e y a imagem).
- Minimiza o conceito de "variável" global, substituindo-o por variável local, logo apresentando *transparência referencial*. Isso diminui de maneira quase inacreditável os erros de programa.

[1] Definição alternativa: LISP = Lot of Idiots and Stupids Parenthesis

- minimiza o uso de atribuições. Na programação funcional original não há a atribuição, mas claro que isso é impraticável.
- Não possuem mecanismos de iteração, substituindo-os por recursão ou por mecanismos específicos à linguagem.
- ciclo leia-calcule-imprima. Em geral o computador se comporta como uma (super) calculadora de mão. Isso aumenta a produtividade ao programar, pois a impressão de resultados já está pronta.
- funções de ordem superior: estas são funções que admitem outras funções (primitivas ou do usuário) como parâmetros da função.
- fracamente tipada (LISP só tem átomos e listas, APL redefine tipos ao bel prazer do programador). Em LISP valores (e não variáveis) é que tem tipo. Objetos têm mais de um tipo. Assim, por exemplo o número `140` é do tipo: `fixnum`, `integer`, `rational`, `real`, `number`, `atom`, e `t`.
- conjunto pequeno (mas poderoso) de primitivas
- capacidade de processamento paralelo
- programas e dados são representados igual e armazenados igual também.
- programas são menores e apresentam menos erros. Esta última não é uma característica formal da programação funcional mas é uma conseqüência real do seu uso. A programação funcional permite a *depuração iterativa*. Funções podem ser testadas imediatamente após prontas. Se elas retornam o valor esperado, pode-se confiar nelas. Essa confiança crescente, faz uma enorme diferença ao final.

Os programas trabalham devolvendo valores e não modificando coisas. As funções são chamadas pelos valores que elas retornam, não por seus efeitos colaterais. A função `remove`, por exemplo, toma uma lista e um valor e devolve outra lista na qual o valor foi excluido.

```
> (setf x '(c u r i t i b a))
(C U R I T I B A)
> (remove 'i x)
(C U R T B A)
```

Não se pode dizer que `remove` remove os elementos da lista, pois a lista original permanece intocada.

```
> x
(C U R I T I B A)
```

E, se realmente se quiser eliminar os elementos da lista ? Seria necessário algo como

```
> (setf x (remove 'i x))
```

A programação funcional tenta evitar assinalamentos, como se verá a seguir.

Compare-se uma função LISP com a sua correspondente C

```
; LISP
(defun sum (n)
  (let ((s 0))
  (dotimes (i n s)
    (incf s i))))
```

```
/* C */
int sum(int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    s = s + i;
  return(s);
}
```

Seja agora um exemplo de uma função lisp:

```
> (defun li (L1 L2)   ; pergunta se duas listas são iguais
    (cond
       ((null L1) (null L2))
       ((null L2) nil)
       ((not (eql (car L1) (car L2))) nil)
       (t (li (cdr L1) (cdr L2)))))
LI
(defun h(x) ; esta funcao e a estimadora do tarkin
   (setq qtd 0) ; quantidade de concordancias
   (setq obje '(1 2 3 8 9 4 7 6 5))
         ; objetivo final do tabuleiro (9 e branco)
   (mapcar '(lambda (y z)
      (cond ((eql y z) (setq qtd (+ qtd 1))))
         ; compara e se igual qtd++
      ) x obje)
   (- 9 qtd)) ; o retorno e' 9 - qtdade de concordancias
```

Chama a atenção que linguagens funcionais (ou pelo menos LISP e APL) foram construídas primeiro como ferramentas de notação, e só depois, muito depois, é que foram implementadas em computador. De ruim, este fato trouxe a lerdeza das primeiras implementações, mas de bom, ele trouxe uma pureza notacional que inexiste nas linguagens há por aí.

Por exemplo, a construção $A = A + 1$ de C, COBOL, PASCAL, Java, ... conceitualmente falando é uma insanidade. É muito melhor a notação APL $A \leftarrow A + 1$. Outro exemplo:

Por reproduzirem a notação humana, a maioria das linguagens aceitam a simplicidade e a facilidade em prejuízo de maior rigor. Por exemplo, A função unária "oposto" é escrita como -1, a função binária "adição" é escrita como $1 + 2$, e como escrever uma função trinária ? (por exemplo x^y mod n). Neste caso, as linguagens precisam definir uma nova função XEYMN(3,4,5). Veja-se agora a pureza notacional de LISP: $(-\ 1), (+\ 1\ 2), (xyn\ 3\ 4\ 5)$.

7.1.2 História do LISP

O LISP nasce como filho da comunidade de Inteligência Artificial. Na origem, 4 grupos de pessoas se inseriram na comunidade de IA: os lingüistas (na busca de um tradutor universal), os psicólogos (na tentativa de entender e estudar processos cerebrais humanos), matemáticos (a mecanização de aspectos da matemática, por exemplo a demonstração de teoremas) e os informáticos (que buscavam expandir os limites da ciência).

O marco inicial é o encontro no Dartmouth College em 1956, que trouxe duas conseqüências importantes: a atribuição do nome (IA) e a possibilidade dos diferentes pesquisadores se conhecerem e terem suas pesquisas beneficiadas por uma interfecundação intelectual.

 ©88-09, Pedro Kantek

A primeira constatação foi a de que linguagens existentes privilegiavam dados numéricos na forma de arrays. Assim, a busca foi a criação de ambientes que processassem símbolos na forma de listas. A primeira proposta foi IPL (Information Processing Language I, por Newel e Simon em 1956). Era um assemblão com suporte a listas. A IBM engajou-se no esforço, mas tendo gasto muito no projeto FORTRAN decidiu agregar-lhe capacidade de listas, ao invés de criar nova linguagem. Nasceu a FLPL (Fortran List Processing Language).

Em 1958, McCarthy começou a pesquisar requisitos para uma linguagem simbólica. Foram eles: recursão, expressões condicionais, alocação dinâmica de listas, desalocação automática. O FORTRAN não tinha a menor possibilidade de atendê-las e assim McCarthy junto com M. Minsky desenvolveu o LISP. Buscava-se uma função Lisp Universal (como uma máquina de Turing Universal). A primeira versão (chamada LISP pura) era completamente funcional. Mais tarde, LISP foi sofrendo modificações e melhoramentos visando eliminar ineficiências e dificuldades de uso. Acabou por se tornar uma linguagem 100% adequada a qualquer tipo de resolução de problema. Por exemplo, o editor EMACS que é um padrão no mundo UNIX está feito em LISP.

Inúmeros dialetos LISP apareceram, cada um queria apresentar a sua versão como sendo a melhor. Por exemplo, FranzLisp, ZetaLisp, LeLisp, MacLisp, Interlisp, Scheme, T, Nil, Xlisp, Autolisp etc. Finalmente, como maneira de facilitar a comunicação entre todos estes (e outros) ambientes, trabalhou-se na criação de um padrão que foi denominado Common Lisp. Como resultado o Common Lisp ficou grande e um tanto quanto heterogêneo, mas ele herda as melhores práticas de cada um de seus antecessores.

7.1.3 Como obter um LISP

LISPs existem aos montões. Em cada esquina da Internet você encontra algum. Eis os que foram pesquisados para a produção deste material (todos rodando sob Windows ou sob DOS)

COMMON LISP

Roda sob uma janela DOS de qualquer Windows. A referência completa de Common Lisp está em http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/cltl/cltl2.html . Trata-se de um livro de mais de 1000 páginas, disponível em html e ps. O software está em

`http://www.lcmi.ufsc.br/gia/lisp-systems/clisp.zip`

Xlisp

Um pacote bem simples mas que roda em windows (16b e 32b). Pode ser obtido em http://www.almy.us/xlisp.html. Basta descarregar o arquivo Logicamente tudo o que vai ser estudado aqui poderá ser implementado em qualquer um destes ambientes (e em todos os demais, por certo).

7.1.4 Introdução e conceitos iniciais

Lisp é uma linguagem dinâmica (originalmente interpretada) cujos programas são pequenos módulos de funcionalidade genérica e com objetivo restrito. Um programa completo é obtido pela combinação desses módulos.

O conceito básico em LISP é o de FORMA (ou fórmula). Isso porque o LISP é como se fosse (é) uma calculadora, à qual são apresentadas fórmulas e ele as responde. Uma fórmula bem comum é um inteiro, por exemplo 8. Se você entregar a ele a fórmula 8, ele devolve o resultado da fórmula, que é 8. Veja no exemplo

```
> 8    ; um inteiro
8
```

Algumas observações deste trecho de conversa com LISP. O sinal > indica a espera por uma entrada. O que aparece depois do > foi digitado pelo operador. Na linha de baixo a resposta de LISP. O sinal de ";"indica que o que vem depois é um comentário e deve ser desprezado pelo LISP.

Este ciclo de entrar fórmula, o LISP avaliá-la e devolver um resultado é o que caracteriza o trabalho do interpretador. Quando você escrever um programa LISP, o programa nada mais será do que um conjunto de fórmulas que serão processadas uma a uma (como se estivessem sendo fornecidas no teclado) enquanto o programa vai sendo executado. Existe um nome técnico para este comportamento que é o ciclo eval-apply-print .

7.1.5 Funções

Uma parte importante das fórmulas são as funções. As mais simples que conhecemos são as aritméticas (+, -, *, /, ...). Vamos considerar cada função como uma caixa que recebe o(s) seu(s) operando(s) e devolve o resultado esperado. Por exemplo, veja uma função adição usual, na figura 7.1 Neste exemplo, verifica-se que a função + recebe 2

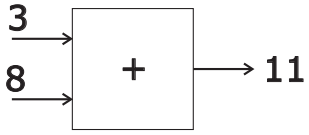


Figura 7.1: Um exemplo da função adição

operandos, no caso o 3 e o 8, e devolve o resultado de 3+8=11. Note que para algumas funções, a ordem de entrada é importante, por exemplo, na figura 7.2.

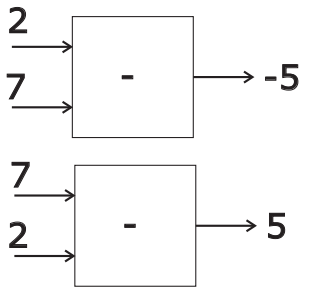


Figura 7.2: A ordem dos operandos é importante

A função / devolve a parte inteira da divisão dos operandos. Assim, 3/5 é 0, 5/3 é 1, 6/3 é 2 e assim por diante.

Quando fornecemos ao LISP uma expressão, por exemplo 2 mais 2, estamos pedindo a ele que a avalie esta fórmula. Ele faz isso chamando a função mais e entregando a ela os 2 operandos. Tudo funciona como se a função fosse uma caixa similar a essa aí de cima.

7.1.6 Símbolos e Números

Um número em LISP é qualquer conjunto de dígitos (de 0 a 9), opcionalmente com um sinal (- ou +) e opcionalmente com um ponto separador. Números dividem-se em inteiros

 ©88-09, Pedro Kantek

e fracionários. Inteiros são entrados como uma seqüência de números opcionalmente precedidos do sinal de menos. Não se colocam aspas. Números fracionários podem ser entrados com o ponto decimal ou com uma barra de fração. Por exemplo, 3/4 é a mesma coisa que 0.75.

Uma classe de funções importante em LISP é a dos predicados, que respondem a uma pergunta e têm como resposta t ou nil. Usualmente, seu nome termina em "P".

O predicado INTEGERP devolve t se o operando é inteiro, nil senão. O predicado NUMBERP devolve t se o operando é número (seja inteiro ou fracionário). O predicado ATOM devolve t se o operando é símbolo ou número, nil senão.

Outro símbolo importante em LISP são as palavras. Servem para nomear variáveis e funções, são usados como dados também. Constróem-se com quaisquer letra, podem ter números, mas sem começar por eles, e também o sinal de hífen.

Finalmente, números e símbolos podem ser considerados como ÁTOMOS.

Existem 4 atributos associados a um símbolo

1. seu nome (um string qualquer)
2. valor corrente: pode ser qualquer dado lisp. O valor default para um valor é o próprio nome do símbolo
3. lista de propriedades (default NIL)
4. função associada: aplicada aos operandos quando este símbolo é chamado como uma função.

A função SYMBOLP retorna t se o operando for um símbolo e nil senão. (symbolp ()) é T, e (symbolp NIL) também é T. Quando quisermos criar um determinado valor corrente para um string que tenha brancos, parênteses ou caracteres especiais, devemos colocar todo o string entre aspas duplas A função SET serve para setar um valor dentro de um símbolo. Exemplo:

```
(set 'vogais '(a e i o u)). ;Escrevendo agora
> vogais  ; <enter>,
a resposta é (a e i o u).
```

Já que o primeiro argumento de SET sempre precisa ser precedido de uma aspas, existe a função SETQ que prescinde da primeira aspas, MAS NÃO DA SEGUNDA. É uma maneira mais econômica de escrever o comando SET. Exemplo:

```
(setq vogais '(a e i o u)).
```

Uma variável pode ser usada mais de uma vez em uma única linha. Vale sempre, para efeito de avaliação da fórmula, a hierarquia determinada pelos parênteses. Por exemplo:

```
(setq vogais (cons 'y vogais)).
```

Símbolos t e nil

Existem 2 símbolos especiais em LISP que atendem pelo nome de t (de TRUE) e nil (de vazio). São pela própria definição, sinônimos de sim e de não. Ou também de Verdade e de Falso. Como já vimos, toda função LISP que tem como resposta um t ou um NIL, recebe o nome de predicado. Portanto, um predicado é uma função LISP que responde uma pergunta (com sim ou não). Boa parte dos predicados em LISP tem um nome terminado pela letra P. Por exemplo, NUMBERP, SYMBOLP, ZEROP, ODDP, que funcionam como mostrado na figura 7.3 Então A resposta de um predicado verdadeiro é t, de um falso é nil e qualquer coisa diferente de nil pode ser entendida como t. Por

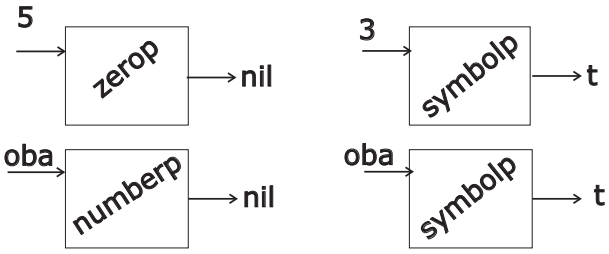


Figura 7.3: Predicados e os símbolos t e nil

exemplo, se eu disser que a resposta de ZEROP(0) é 8, isto formalmente não está errado, pois 8 não é nil, logo 8 pode ser entendido como t (quando resposta a um predicado, é óbvio).

Outro predicado importante é o NOT, que inverte o valor lógico da entrada. Assim NOT de nil é t e NOT de t é nil. Na verdade, o NOT de qualquer coisa (exceto t) é t.

Existe também o predicado EQUAL que devolve t se seus dois operandos são exatamente iguais e nil em caso contrário.

O predicado ODDP (ímpar) devolve t se o seu operando por ímpar e nil senão. O predicado EVENP (par) devolve t se o operando por par e nil senão.

Funções Compostas

Vamos começar definindo a função SOMAUM, que poderia ter a estrutura definida na figura 7.4. Com SOMAUM definido, podemos construir SOMADOIS, conforme a figura

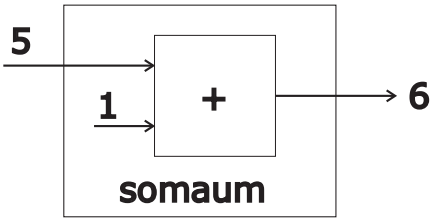


Figura 7.4: Uma função específica de soma um

7.5. Usando o critério de encaixar caixas, podemos construir funções cada vez mais ricas.

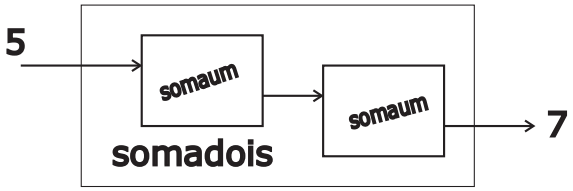


Figura 7.5: Uma função composta

Se quisermos somar dois números, por exemplo 4 e 5, deveremos escrevem em LISP (+ 4 5) ao invés do nosso usual 4+5. Isto porque a fórmula esperada pelo LISP é uma lista formada por "abre parênteses", operador, operandos... e "fecha parênteses".

Se quisermos calcular 2 + 4 + 6, em LISP deveremos escrever (+ 2 4 6). É a chamada notação pré-fixada. Note que na aritmética usual, cada sinal de "+"exige 2 operadores, um à esquerda e um à direita. Em LISP podemos ter qualquer número de operadores:

 ©88-09, Pedro Kantek

```
>(+)
0
>(+ 2)
2
>(+ 3 7)
10
>(+3 6 9)
18
>(+ 1 2 3 4 5)
15
```

Nada impede que você construa funções compostas, usando várias listas, por exemplo

```
>(+(-7 2) (+2 3)) ; é igual a
10
```

A sequencia de execução do comando acima é:
a) avalia-se - 7 2, que dá como resposta 5
b) avalia-se + 2 3, que dá como resposta 5
c) avalia-se + 5 5, que dá como resposta 10.

EXERCÍCIO 1 Usando as funções +, -, *, / e os predicados EQUAL, NOT, ZEROP, >, <, ODDP, EVENP, escreva as caixas que determinam as seguintes funções:

1. MAIORQUEUMP, que devolve t se o primeiro operando é igual ao segundo operando mais um e nil senão.

```
> (defun mq (x y)
   (if (> x (+ y 1)) t nil))
MQ
> (mq 1 9)
NIL
> (mq 9 1)
T
> (mq 1 2)
NIL
> (mq 2 1)
NIL
> (mq 3 1)
T
```

2. METADE que devolve a metade do operando fornecido.

```
> (defun mt (x)
(/ x 2.0))
MT
```

3. POSITIVOP, que devolve t se o operando por maior ou igual a zero.

```
> (defun pos (x)
    (or (> x 0) (= x 0)))
POS
> (pos 5)
```

```
T
> (pos 0)
T
> (pos -1)
NIL
```

4. DIFERENTEP, que devolva t se os dois operandos forem diferentes (não iguais) e nil senão.

```
> (defun diferentep (x y)
    (not (= x y)))
DIFERENTEP
> (diferentep 5 5)
NIL
> (diferentep 5 4)
T
```

5. MAIOROUIGUALP, que devolva t se o primeiro operando é maior ou igual ao segundo operando e nil senão.

6. MARIAP, que devolve t se o seu operando for MARIA e nil senão.

```
> (defun mariap (x)
    (equal x 'maria))
MARIAP
> (mariap 'maria)
T
> (mariap 'jose)
NIL
```

7. SOMA3 que receba 3 valores e devolva a sua soma.

```
> (defun soma (x y z)
    (+ x y z))
SOMA
> (soma 1 2 3)
6
```

Domínio e contra-domínio de uma função

O domínio de uma função é o conjunto dos valores possíveis de entrada para a função e seu contra-domínio (ou imagem) é o conjunto de todos os valores possíveis de saída. Assim, o domínio da função ODDP é o conjunto dos números inteiros e sua imagem é o conjunto t, nil.

Define-se o fechamento como uma propriedade que pode se aplicar a alguma função. Uma função é fechada em seu domínio quando os valores de saída podem ser usados como entrada para a mesma função. Por exemplo, a função SOMAUM acima é fechada em seu domínio. A função ZEROP não é.

Função inversa de uma função dada é aquele que ao receber a saida da função dada, produz a entrada original a ela. Por exemplo, a função SOMAUM poderia ter a inversa denominada SUBTRAIUM. Nem toda função tem inversa.

EXERCÍCIO 2 Escreva a notação LISP corresponde a estas operações

 ©88-09, Pedro Kantek

```
5 + 6 - 7
5 - 6 * 7
5 * 6 - 7
5 * 6 * 7
(5 - 6) * 7
(5 - 6) + 7
5 - (6 + 7)
5 * 5 + 6 * 6 * 6
```

```
Resposta:

(- (+ 5 6) 7) ou (+ 5 (- 6 7)) ou (+ 5 6 (- 7))
(- 5 (* 6 7))
(- (* 5 6) 7)
(* (* 5 6) 7) ou (* 5 (* 6 7)) ou (* 5 6 7)
(* (- 5 6) 7)
(+ (- 5 6) 7)
(- 5 (+ 6 7))
(+ (* 5 5) (* 6 6 6)
```

EXERCÍCIO 3 Converta as expressões LISP para notação convencional

```
(* (/ 4 5) 6)
(* 4 (- 5 6))
(/ (+ 4 5) 6)
(/ (/ 4 5) 6)
(/ 4 (/ 5 6))
(- (- 4 5) 6)
(- 4 5 6)
```

```
Resposta

4 / 5 * 6
4 * (5 - 6)
(4 + 5) / 6
4 / 5 / 6
4 / (5 / 6)
(4 - 5) - 6
4 - 5 - 6
```

EXERCÍCIO 4 Calcule o valor das seguintes expressões Lisp:

```
(* (/ 1 2) 3)
(* 1 (- 2 3))
(/ (+ 1 2) 3)
(/ (/ 1 2) 3)
(/ 1 (/ 2 3))
(- (- 1 2) 3)
(- 1 2 3)
(- 7)
```

```
Resposta
```

```
3/2
-1
1
1/6
3/2
-4
-4
-7
```

7.1.7 Listas

Uma lista é uma composição de dados. Neste sentido um dado composto (lista) é o antônimo de átomo. Uma lista é um objeto fundamental em LISP (não esqueça o significado de LISP). Tudo em LISP pode ser representado como uma lista. Inclusive funções são representadas como listas (são listas!) em LISP.

Listas são representadas de duas maneiras, a externa e a interna. Para o usuário de LISP apenas a representação externa interessa, mas para quem quer conhecer LISP e programá-lo, ambas são importantes.

Representação externa

A representação externa de uma lista começa com um "abre parênteses", segue por uma lista de átomos – separados por pelo menos um espaço e terminado por um fecha parênteses.

Exemplos de listas

```
(LA NO ALTO DE TANTAS GLORIAS)
Uma lista com 6 elementos
(22)
Lista com um único elemento, o 22
()
Lista vazia
(2 APERTOS 1 BOLACHA 33 3 21)
Lista com 7 elementos
(A B (C D) E)
Lista com 4 elementos. O terceiro, por
sua vez é uma lista de 2 elementos
```

Este último exemplo, merece algum cuidado. Perceba a diferença entre a lista (A B (C D) E) e a lista (A B C D E). A primeira tem 4 elementos e a segunda 5. Esta construção permite a elaboração de listas mais sofisticadas, por exemplo, suponha uma lista de pessoas e para cada pessoa queremos ter o nome, a idade e a cidade de nascimento. Poderia ficar

```
((Paulo 27 Curitiba) (Marília 23 Florianópolis) (José 45 Maringá))
```

Quem se sentir desconfortável com esta representação pode usar:

```
((Paulo 27 Curitiba)
 (Marília 23 Florianópolis)
 (José 45 Maringá))
```

 ©88-09, Pedro Kantek

É a mesma coisa. Outra observação a ser feita neste ponto é que () é para todos os efeitos sinônimo de nil. Embora nil seja um símbolo e () uma lista eles significam a mesma coisa e são intercambiáveis. Portanto (A nil B) é a mesma lista que (A () B).

Note que necessariamente deverá haver uma correção no número e na disposição dos parênteses na fórmula LISP. Verifique os seguintes erros possíveis

```
(LA NO ALTO (DE TANTAS GLORIAS)
dois abre parêntes e só um fecha
)22(
Primeiro fecha e depois abre
())
Um abre e dois fecha
(A B C
abre e não fecha
```

EXERCÍCIO 5 Para cada uma das listas a seguir, responda:

1. quantos elementos tem a lista
2. qual é o seu primeiro elemento
3. qual é o seu último elemento

```
a. (((A B) C ((D E F) G H) K) L M)

b. (A B C (D E F) (G H) (I J))

c. ((A B) (C D) (E))

d. (((A)))

e. (A B C D (E F G))
```

Representação interna

Os elementos de LISP existem em um ambiente onde há uma tabela de símbolos e também um "saco"de células CONS. As células CONS se unem com o auxílio da tabela de símbolos para formar as listas, que como vimos são o elemento fundamental em LISP.

Uma célula CONS é um conjunto de 2 endereços que apontam para a tabela de símbolos. Elementos da tabela, por sua vez apontam para células CONS que são encadeadas.

O primeiro endereço da célula CONS é chamado CAR e o segundo é chamada CDR. A origem dos nomes é histórica. Na primeira implementação de LISP ainda na década de 50, usou-se um computador IBM 704 e a célula CONS foi construída usando dois dos registradores da máquina (CAR = Contents of the Address part of Register e CDR = Contents of the Decrement part of Register). COMMON LISP possui os sinônimos "first"e "rest"mas parece que ninguém os usa.

Neste desenho vemos o primeiro endereço como sendo o CAR da célula CONS e o segundo como sendo o endereço CDR da mesma célula.

Suponhamos agora querer entender a representação interna da lista

```
(TENHO 22 REAIS)
```

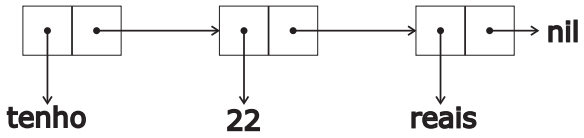


Figura 7.6: A lista (TENHO 22 REAIS) na memória

Os endereços em cada célula CONS referem-se a entradas na tabela de símbolos onde de fato os átomos ("TENHO", 22 e "REAIS") foram definidos. nil refere-se a uma das entradas da tabela que contém nil.

Agora suponhamos a representação interna da seguinte lista

```
(A (B C) D)
```

Eis como ficaria na memória da máquina

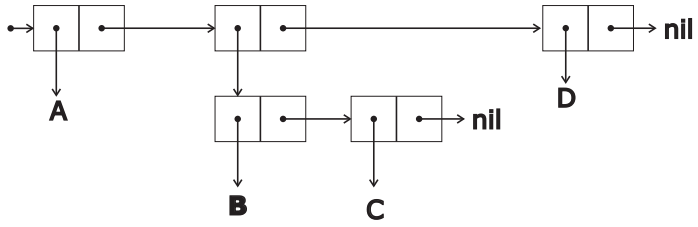


Figura 7.7: A lista (A (B C) D) na memória

EXERCÍCIO 6 Escreva a representação interna das seguintes listas:
1. ((ALBERTO E O) GATO AZUL)
2. (UM DIA (A CASA (AMARELA) VAI) CAIR)
3. (((AÇUCAR)))

EXERCÍCIO RESOLVIDO 1 Exercício sobre alocação de memória pelo interpretador LISP:
Se imaginarmos que a tabela de símbolos começa no endereço 1000 (tamanho de cada entrada=12) e o "saco"de CONS comeca em 2000, (tamanho de cada CONS=8), poderíamos ter a seguinte situação:

```
ENDE-CONT
1000-0580
1012-0639
1024-1057
1036-3358
1048-3777
1060-4155
1072-4450
1084-4466
1096-JTYA
1108-KJFC
2000-L1
1132-OYGZ
```

 ©88-09, Pedro Kantek

```
1144-QBDF
1156-RZZH
1168-UVKJ
1180-nil
1192-t

ENDE  CAR  CDR
2000- 2008 2016
2008- 1060 1180
2016- 1048 2024
2024- 2032 2040
2032- 1168 1180
2040- 1024 2048
2048- 1156 2056
2056- 2064 2088
2064- 2072 2080
2072- 1108 1180
2080- 1072 1180
2088- 2096 2152
2096- 2104 2112
2104- 1144 1180
2112- 2120 2128
2120- 1012 1180
2128- 1036 2136
2136- 1084 2144
2144- 1000 1180
2152- 2160 1180
2160- 1096 2168
2168- 1132 1180
```

Este exercício terá como resposta:
((4155) 3777 (UVKJ) 1057 RZZH ((KJFC) 4450)((QBDF) (0639) 3358 4466 0580) (JTYA OYGZ))

Exercício 7

```
1.        TABELA DE SÍMBOLOS
            ENDE CONTEÚDO
            1000-0229
            1012-1331
            1024-2464
            1036-2782
            1048-3277
            1060-3747
            1072-3948
            1084-BBQH
            1096-CQNZ
            1108-JCRI
            2000-L1
            1132-LVOX
            1144-UWYC
            1156-WRWU
            1168-ZLTD
            1180-nil
```

```
            1192-t

          SACO DE CONS
        ENDE  CAR  CDR
        2000- 1096 2008
        2008- 1132 2016
        2016- 1168 2024
        2024- 2032 2040
        2032- 1156 1180
        2040- 2048 2056
        2048- 1036 1180
        2056- 1108 2064
        2064- 2072 2120
        2072- 2080 2088
        2080- 1048 1180
        2088- 1024 2096
        2096- 2104 2112
        2104- 1084 1180
        2112- 1144 1180
        2120- 1012 2128
        2128- 1000 2136
        2136- 1072 2144
        2144- 2152 1180
        2152- 2160 1180
        2160- 1060 1180
```

A representação externa da lista solicitada é:
L1=_____.

```
2.       TABELA DE SÍMBOLOS
            ENDE CONT
            1000-0371
            1012-0895
            1024-1379
            1036-1417
            1048-1529
            1060-3333
            1072-4343
            1084-BYEQ
            1096-GLRO
            1108-KIGO
            2000-L1
            1132-MHDF
            1144-SOYJ
            1156-VDZL
            1168-WIYY
            1180-nil
            1192-t

         SACO DE CONS
        ENDE  CAR  CDR
        2000- 1168 2008
        2008- 2016 2024
```

 ©88-09, Pedro Kantek

```
2016- 1012 1180
2024- 2032 1180
2032- 1084 2040
2040- 2048 2056
2048- 1156 1180
2056- 1072 2064
2064- 2072 1180
2072- 2080 2088
2080- 1024 1180
2088- 2096 1180
2096- 2104 2120
2104- 1132 2112
2112- 1096 1180
2120- 1048 2128
2128- 2136 2144
2136- 1060 1180
2144- 2152 2160
2152- 1144 1180
2160- 2168 2176
2168- 1036 1180
2176- 2184 1180
2184- 2192 2200
2192- 1108 1180
2200- 1000 1180
```

A representação externa da lista solicitada é:
L2=_____.

```
Respostas
1:  (CQNZ LVOX ZLTD (WRWU) (2782) JCRI ((3277) 2464 (BBQH)
    UWYC) 1331 0229 3948 ((3747)))

2:  (WIYY (0895) (BYEQ (VDZL) 4343 ((1379) ((MHDF GLRO)
    1529 (3333) (SOYJ) (1417) ((KIGO) 0371)))))
```

Desafio

- Localize na internet um LISP que seja freeware e instale-o em seu computador
- Entre com os comandos e funções estudados na aula e verifique o comportamento do compilador nesta tarefa.

7.2 Funções CAR e CDR

Já se viu o que são as partes CAR e CDR de cada célula LISP. Vamos estudar agora as funções CAR e CDR de uma lista. Em outras palavras, quando se solicita o CAR de

Figura 7.8: CAR e CDR de listas

uma lista, o LISP devolve o primeiro elemento desta lista. Já o CDR retorna a lista que fica quando o primeiro elemento (apontado pelo CAR) é dela retirado.

Acompanhe nos exemplos
CAR de ((AZUL VERDE ROSA) (MORANGO AÇUCAR CARAMELO)) é (AZUL VERDE ROSA)
CAR de ((COXA BRANCA)) é (COXA BRANCA)
CAR de OVO é erro, pois "OVO"não é uma lista, e sim um átomo.
CDR de ((AZUL VERDE ROSA) (MORANGO AÇUCAR UVA)) é ((MORANGO AÇUCAR UVA))
CDR de ((COXA BRANCA)) é nil
CDR de OVO é erro.

Atente que tanto o CAR quanto o CDR de nil são nil. Existe uma função chamada LENGTH que retorna o número de elementos da lista fornecida. Assim:

```
> (length '(A B C D E)) ; é
5
> (length '(A (B C D) E) ; é
3
> (length '((A B C D E)) ; é
1
```

7.2.1 Representação de funções LISP

Como tudo em LISP é uma lista, assim também são as funções: Ao invés de representar as funções através de caixas – como temos feito até agora – vamos usar a notação EVAL, na qual funções são representadas na forma de listas. Assim, uma função LISP é uma lista (logo entre parênteses) contendo como primeiro elemento, o nome da função, e como demais elementos os operandos da função. Embora não tenhamos dito isso na ocasião, eis como se representam em LISP algumas funções aritméticas:

Em aritmética usual	em LISP
2 + 3	(+ 2 3)
5 / 2	(/ 5 2) - observe a ordem dos elementos
5 > 2	(> 5 2)

A mesma regra vale para todas as funções LISP, assim

```
(CAR (A B C D)) é A
(CDR (A B C D)) é (B C D)
(CAR ()) é ()
(CDR ()) é ()
```

A função EVAL é o cérebro de qualquer ambiente LISP. A função EVAL lê funções (como se fossem – e são – dados), avalia-as e se corretas, as executa.

As regras de avaliação que a EVAL utiliza são:

- Um número é avaliado como ele mesmo
- idem para t e nil
- uma lista é avaliada tomando a função CAR da lista e passando a ela o CDR da lista.
- Parênteses determinam a prioridade em funções compostas.

Atente-se que TUDO o que é feito com caixas, pode ser representado pela notação EVAL, por exemplo

 ©88-09, Pedro Kantek

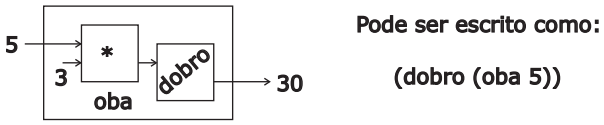


Figura 7.9: Transformando caixas em *eval*

A importância da ASPA

Se escrevemos CONS(t nil) a resposta é (t), já que t é valiado como t e nil como nil. Entretanto, se fizermos CONS(HOJE nil), a resposta poderá ser um erro informando que o LISP desconhece qual o valor da variável HOJE. (Em LISP diz-se que HOJE não está ligada). A maneira de informar ao LISP que um certo valor deve ser entendido como um valor e não como uma variável é pelo uso de uma "aspa".

Em outras palavras, quando na frente de alguma coisa o LISP encontra uma aspa, ele entende essa alguma coisa como um valor. Então CONS('HOJE nil) é (HOJE). Essa regra vale também para listas, então:

(CAR(A B C D)) dará como resposta um erro informando que função A não está definida. Para obter o resultado esperado, deve-se fazer (CAR '(A B C D))

Mais exemplos e contra-exemplos

```
(+ 2 3) é 5 e '(+ 2 3) é (+ 2 3)
'(EU QUERO MOCOTO) é avaliado como (EU QUERO MOCOTO)
(LIST 'EU 'QUERO 'MOCOTO) é avaliado como (EU QUERO MOCOTO)
(CONS 'EU '(QUERO MOCOTO)) é avaliado como (EU QUERO MOCOTO)
(LIST EU QUERO MOCOTO) é avaliado como ERRO: variável EU desconhecida
(EU QUERO MOCOTO) é avaliado como ERRO: função EU desconhecida
('EU 'QUERO 'MOCOTO) é avalidada como ERRO: função 'EU desconhecida
                                  (veja a aspa)
```

7.2.2 Definição de funções

A criação de uma nova função em LISP se dá pela escrita de uma lista. Esta lista deve ter pelo menos 4 coisas, a saber

- A palavra `defun` abreviatura de *define function* e que informa ao LISP que o que segue deve ser armazenado para executar depois.
- Um nome válido que nomeará a função.
- Uma lista (podendo ser até vazia) de parâmetros, indicando com quais operandos esta função será chamada depois.
- um conjunto de tarefas que informam o que e como a função vai fazer alguma coisa

Com esta explicação, fica fácil perceber o formato de uma definição (acompanhe e não se perca com os parênteses):

```
(defun NOME-FUN (arg1 arg2 ...) tarefa1 tarefa2...)
```

O LISP responde a esta definição com o nome da função recém definida.

Seja, por exemplo, a construção de uma função chamada PRIMEIRO que devolva o primeiro elemento de uma lista:

```
> (defun primeiro (lista)
    (car lista))
PRIMEIRO

> (primeiro '(a b c d e))
A
```

Definindo o segundo:

```
(defun segundo (lista)
   (car (cdr lista))
   )
```

O terceiro

```
(defun terceiro (lista)
   (car (cdr (cdr lista)))
   )
```

Defindo função que volte t se o operando é nulo, ou nil senão

```
> (defun nulo (obj)    ; esta função retorna
    (eql nil obj))     ; t se o operando é nulo
NULO
> (nulo ())
T
> (nulo 'oba)
NIL
```

Se quisermos definir uma função que não tenha parâmetros de entrada, na definição do cabeçalho deve-se colocar uma lista vazia. Exemplo:

```
(defun SAUDA ()
   'BOA 'TARDE)
```

A chamada desta função deverá ser

```
>(SAUDA)
BOA TARDE
```

Regra de Avaliação de uma função definida

O corpo de uma função do usuário (um programa) é um conjunto de tarefas. Cada tarefa não condicional vai sendo executada normalmente. Quando chegar uma tarefa condicional (COND), espera-se que as suas sub-tarefas tenham como CAR um predicado. Nestas, vale a regra:

- Se o predicado retorna NIL esta sub-tarefa não é executada e o fluxo segue para a próxima sub-tarefa.
- Se o predicado retorna algo diferente de NIL as sub-tarefas remanescentes são ignoradas e o fluxo segue para a próxima tarefa.

Exemplo: escrever uma função que devolva t se seu operando for uma lista

```
> (defun listaa (oo)
    (cond
      ((atom oo) nil)
      (t t)))
```

 ©88-09, Pedro Kantek

```
LISTAA
> (listaa 'a)
NIL
> (listaa '(1 2))
T
> (listaa '(1))
T
```

Ver se um nome pertence a uma lista

```
> (defun xan (nome lista)
    (cond
      ((null lista) nil)
      ((equal nome (car lista)) t)
      (t (xan nome (cdr lista)))))
XAN
> (xan 'ola '(oba oka ola opa oua))
T
> (xan 'oza '(oba oka ola opa oua))
NIL
```

Ver se duas listas são iguais

```
> (defun li (x y)
    (cond
      ((null x) (null y))
      ((null y) nil)
      ((equal (car x) (car y))(li (cdr x) (cdr y)))
    )
  )
LI
> (li '(1 2 3) '(1 2 3))
T
> (li '(1 2 3) '(1 2 3 4))
NIL
> (li () '(1 2))
NIL
> (li '(1) ())
NIL
```

Definição de não

```
(defun não (objeto)         ; negando objeto
   (eql objeto NIL)         ; se objeto = NIL, devolva T; senão NIL
   )
```

Note-se que a definição de não é igual a definição de nulo. Isto porque o LISP usa nil tanto como valor FALSO como lista vazia.

Função CONS

A função CONS (de CONStrutora) é usada para construir listas novas. Em geral, CONS acrescenta um elemento no ínicio de uma lista, veja-se no exemplo

Pode-se definir uma função ELOGIO que teria a seguinte formação

Repare que em termos internos, a função CONS cria uma nova célula colocando nela o endereço do primeiro operando e na segunda o endereço do segundo operando.

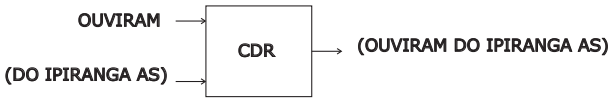


Figura 7.10: Construindo listas com CONS

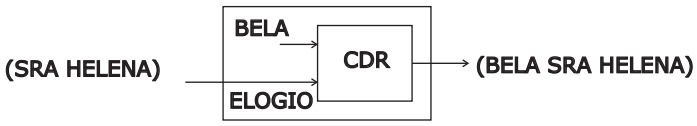


Figura 7.11: A função ELOGIO

```
Acompanhe nos exemplos:
(CONS A (B C D)) é (A B C D)
```

Ligamento de Variáveis

Uma variável ligada é um nome ao qual está associado um valor. Por exemplo se RAIO tiver o valor 3, poderá se dizer que RAIO está ligado ao valor 3. Toda vez em que aparecer a palavra RAIO em uma computação, o valor 3 será usado no lugar de RAIO. (Ó óbvio que ao invés de RAIO aparecer 'RAIO em uma computação é o valor "RAIO"(e não 3) que será usado, por causa da aspa.

Uma primeira modalidade de ligamento de varíavel aparece quando uma função é definida. Atente:

```
(defun PRIMEIRO  (LISTA)
   CAR(LISTA)
   )
```

Uma vez definida a função PRIMEIRO, ela pode ser chamada com (PRIMEIRO '(A B C D)) e a resposta como sabemos será A. Quando a função PRIMEIRO começa, a variável LISTA é ligada com '(A B C D) e quando a função termina, o ligamento é desfeito. Tem-se portanto, um ligamento temporário.

7.2.3 Usando um computador

Qualquer interpretador LISP funciona em um ciclo infinito de leitura, avaliação e impressão. Nesse sentido, ele funciona como se fosse uma calculadora de bolso. Em geral há um símbolo que o interpretador imprime informando estar apto a receber comandos. Em MULISP é um "$", em MacLisp é um "*"e em common lisp é um ">". Todos eles significam uma mensagem de PROMPT. Em geral, o que é escrito pelo operador aparece em minúsculo enquanto a resposta do LISP é em maiúsculo.

Construindo listas

Seja agora a junção de duas listas formando uma única. Lembrando que se usarmos CONS direto o resultado poderá não ser o desejado... exemplo (cons '(maria jose joao) '(alberto paulo xico)) não da uma lista de 6 átomos, e sim uma lista de 4, onde o primeiro é uma lista de 3 ((maria jose joao) alberto paulo xico) em vez de (maria jose joao alberto paulo xico). Outra hipótese é usar LIST. Esta pode receber qualquer número de operandos e devolve uma lista que os contém a todos. Como fazer ? Vejamos o exemplo

 ©88-09, Pedro Kantek

```
> (defun ape (l1 l2)
    (if (null l1) l2
    (cons (car l1) (ape (cdr l1) l2))))
APE
> (ape '(1 2 3 4) '(a b c))
(1 2 3 4 A B C)
> (ape '(1 2 3 4) 'a)
(1 2 3 4 . A)
```

Note que existe uma função APPEND, igual à aqui definida que já é primitiva do LISP. Para ajudar na fixação, perceba-se a diferença entre APPEND, LIST e CONS:

```
>(APPEND '(A B C D) '(E F G H)) é (A B C D E F G H)
>(LIST '(A B C D) '(E F G H)) é ((A B C D) (E F G H))
>(CONS '(A B C D) '(E F G H)) é ((A B C D) E F G H)
```

Seja agora a seguinte questão: Elaborar uma função chamada "remo" que elimina a *primeira* ocorrência do átomo (1. operando) na lista (2. op).

```
> (defun remo (ato lis)
    (cond
      ((null lis) nil )
      ((eql ato (car lis)) (cdr lis))
      (T (cons (car lis) (remo ato (cdr lis))))))
REMO
> (remo '1 '(1 2 3 4 5))
(2 3 4 5)
> (remo '4 '(1 2 3 4 5))
(1 2 3 5)
> (remo '2 '(1 2 2 2 3 2 4 2))
(1 2 2 3 2 4 2)
```

Se o negócio agora for remover TODAS as ocorrências de ato em lis, fica

```
> (defun remo2 (ato lis)
    (cond
      ((null lis) nil )
      ((eql ato (car lis)) (remo2 ato (cdr lis)))
      (T (cons (car lis) (remo2 ato (cdr lis))))))
REMO2

> (remo2 '2 '(1 2 2 2 3 2 4 2))
(1 3 4)
> (remo2 '9 '(1 2 3 4))
(1 2 3 4)
```

Vamos usar agora o seu QR (quociente de recursão). Escreva uma função que receba a lista (vaca cavalo gato cachorro) e devolva (cachorro gato cavalo vaca)

```
> (defun rev (li)
    (cond
      ((null li) nil)
      (t (append (rev (cdr li)) (cons (car li) nil)))))
REV
> (rev '(cao gato vaca carro))
```

```
(CARRO VACA GATO CAO)
> (rev '(1))
(1)
```

Nota: `(cons (car lista) NIL)` é só para deixar `(car lista)` uma lista. A definição acima não é muito eficiente, pelas sucessivas chamadas a append. Para resolver podemos bolar a reverte2, que faz a mesma coisa. Note-se que no cabeçalho de reverte2 vamos citar 2 listas. Se na chamada fornecermos apenas 1, a segunda terá assinalado o valor NIL. *Isto não é verdade em Lispworks. Aqui, a segunda lista tem que ser oferecida explicitamente como vazia.*

```
> (defun reverte2 (t1 t2)   ; feito em lispworks
  (cond
    ((null t1) t2)
    (t (reverte2 (cdr t1) (cons (car t1) t2)))))
REVERTE2
> (reverte2 '(1 2 3 4) () )
(4 3 2 1)
```

Este esquema é igual a inverter uma pilha de cartas na mesa. Vai-se passando de uma pilha para a outra até acabar. Quando acabar a primeira pilha, voilá, está invertida.

A seguir, três maneiras de se construir a mesma lista:

```
a) '(UM DOIS TRES QUATRO) é (UM DOIS TRES QUATRO)
b) (list 'UM 'DOIS 'TRES 'QUATRO) é (UM DOIS TRES QUATRO)
c) (cons 'UM '(DOIS TRES QUATRO)) é (UM DOIS TRES QUATRO)
```

Existe um predicado, de nome LISTP que devolve t se seu operando for uma lista e nil senão. Existe também outro predicado CONSP que devolve t se seu operando é uma lista com ao menos uma célula CONS. Em outras palavras, CONSP devolve t para qualquer lista, EXCETO (). Uma possível definição para CONSP é

```
>(defun CONSPP (X) (AND (LISTP X) (NOT (ATOM X))))
```

EXERCÍCIO 8 Procure a descrição das seguintes funções / predicados:

função	o que faz	parâmetros	devolve
NCONS			
LAST			
NTHCDR			
NTH			
SUBST			
UNION			
INTERSECTION			
SET-DIFERENCE			
MEMBER			
ASSOC			
SUBLIS			
RPLACA			
RPLACD			
DISPLACE			
NCONC			

Existe uma primitiva dentro do LISP chamada LAST que devolve o último elemento de uma lista, como sendo uma lista. Por exemplo

 ©88-09, Pedro Kantek

```
> (last '(1 2 3 4 5))
(5)
```

Para obter o último elemento como elemento, faríamos

```
> (defun ultele (lista)
(car (last lista))
   )
```

EXERCÍCIO 9 O que dá ?
>(setq nome 'joao)
>(setq endereco '(rua das aboboras 23))
>(list nome endereco)
Resposta
(joao rua das aboboras 23)

EXERCÍCIO 10 O que ocorre ?
> (CONS '(DOG CAT) '(COW PIG))
> (LIST '(DOG CAT) '(COW PIG))
> (APPEND '(DOG CAT) '(COW PIG))
Respostas
((DOG CAT) COW PIG)
((DOG CAT) (COW PIG))
(DOG CAT COW PIG)

7.2.4 Aritmética

Números inteiros não tem limite de comprimento. Números fracionários estão limitados a 7 casas decimais de precisão. Como é programação funcional, escreve-se antes o operador e depois os operandos. É a notação polonesa.

O fatorial de um número não negativo e inteiro N, denotado por N!, pode ser definido recursivamente como:

```
0! = 1,
N! = N*(N-1)! if N > 0

> (defun fat (n)
  (cond
    ((equal N 0) 1)
    (t (* N (fat (- N 1))))))
FAT
> (fat 5)
120
> (fat 4)
24
> (fat 100)
933262154439441526816992388562667004907159682643816214685929
638952175999932299156089414639761565182862536979208272237582
511852109168640000000000000000000000000
```

A seqüência deFibonacci , pode ser definida recursivamente como:

```
  F(0) = 1,
  F(1) = 1,
  F(N) = F(N-1) + F(N-2)  if  N > 1.

> (defun FIBONACCI (N)
    (cond
      ((zerop N) 1)
      ((eql N 1) 1)
      (t (+ (FIBONACCI (- N 1)) (FIBONACCI (- N 2))))))
FIBONACCI
> (fibonacci 5)
8
> (fibonacci 3)   ; note que f(0)=1
3
```

Este algoritmo é tremendamente ineficiente (porque ?). Uma versão bem mais eficiente ainda que também recursiva (porque ?) pode ser

```
> (defun FIB (N F1 F2)
    (cond
      ((ZEROP N) F1)
      (t (FIB (- N 1) (+ F1 F2) F1))))
FIB
> (fib 3 1 0)
3
> (fib 10 1 0)
89
```

Encontrar a m-ésima potência de N pode ser resolvido através de uma versão recursiva

```
> (defun POWER (N M)
    (cond
      ((ZEROP M) 1)
      ((EVENP M) (POWER (* N N) (/ M 2))  )
      (t (* N (POWER N (- M 1))))))
POWER
> (power 2 3)
8
> (power 3 2)
9
> (POWER 10 100)
10000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
```

O máximo divisor comum (GCD em inglês) de 2 inteiros é o maior inteiro não negativo que divide ambos sem deixar resto. O algoritmo de Euclides é:
1. Se N = 0, então GCD (N, M) = M;
2. Senão, GCD (N, M) = GCD (M mod N, N).

A definição recursiva em LISP é:

```
> (defun GCD (N M)
((ZEROP N) M)
(GCD (REM M N) N)
```

 ©88-09, Pedro Kantek

```
)
> (GCD 21 56)
7
```

As funções GCD e LCM (mínimo múltiplo comum) são primitivas em LISP. Finalmente, existem as funções =, /=, <, >, >= e <=. Veja:

```
> (= 34 34.0)
T                                ;Equal test
> (/= 3/4 0.75)
nil                              ;Not equal test
> (< 67 45)
nil                              ;Less than test
> (>= 19 17 17)
T                                ;Greater than or equal test
```

Para comodidade do usuário, + contém a última expressão pedida, ++ a penúltima e +++ a ante-penúltima. Já os resultados estão em * (último), ** (penúltimo) e *** (ante-penúltimo). As F1, F2 e F3 funcionam aparentemente no mesmo padrão do DOS (F1 = 1 caracter, F3 = toda a ultima linha).

7.3 Formas condicionais

O comando IF tem três argumentos: uma condição, o que é executado se a condição for t e o que é realizado se a condição for nil. Exemplos:

```
>(IF (> 1 0) 'MAIOR 'NÃO) ;é
MAIOR
>(IF t 1 2) ;é
1
>(IF (not t) 1 2) ;é
2
```

Se o comando IF tiver apenas 2 entradas ao invés de 3 entradas, será considerada a existência virtual da terceira entrada valendo nil.

Estude agora uma função que recebe um número e se ele for positivo deixa-o mais positivo somando uma unidade. Se ele for negativo deve deixá-lo mais negativo substraindo uma unidade. Se for zero, deve deixar a saída como zero.

```
>(defun LONGE (X) (if (zerop X) 0 (if (> X 0) (+ X 1) (- X 1))))
 1          2 2 3   4      4   5   6     6 7     7 8     8531  --> parênteses
```

A função COND recebe um número qualquer de cláusulas condição-ação. Ele examina as cláusulas uma a uma, até que encontre uma que vale t. Quando encontrar, executa a ação correspondente e sai fora do COND pulando as condições-ações remanescentes. Veja-se o exemplo acima usando COND

```
>(defun LONGE1 (X)
(COND ((= X 0) 0)
((> X 0) (+ X 1))
((< X 0) (- X 1)))
```

Note-se que COND e IF são similares. COND é mais flexível ao permitir qualquer número de alternativas, mas sempre é possível construir uma estrutura IF equivalente a uma estrutura COND, como se viu acima no exemplo, usando-se o que se chama de IF aninhado.

É comum também encerrar uma clausula COND com a seguinte expressão (t qualquer-coisa). Note-se que se o LISP chegar até esta cláusula, significa que nenhuma das anteriores foi verdadeira e esta sempre o será. Nesse caso, qualquer-coisa é a clause OTHERWISE para todo o conjunto. (OTHERWISE = se nenhuma das anteriores for verdadeira...)

Vale a penas mencionar aqui as condicionais AND e OR e a função NOT, que tem funcionamento ligeiramente diferente em LISP do que nas demais linguagens:

A regra de AND diz que as cláusulas que a compõem devem ser avaliadas uma a uma. Se uma delas produzir nil, imediatamente há um desvio da condicional AND retornando o que veio. Enquanto não se retornar nil, a AND segue avaliando. Se a última cláusula não retornar nil, a condicional AND retorna o último valor retornado. Exemplos

```
>(AND t t)
t
>(AND (> 1 0) (< 1 0))
nil
>(AND (+ 2 2) (> 1 0))
t
>(AND (> 1 0) (+ 2 2))
4
```

Já a condicional OR faz com que sejam avaliadas as cláusulas, uma a uma. Se uma devolve nil, a condicional prossegue na próxima cláusula. Se a cláusula devolve algo diferente de nil a OR encerra imediatamente devolvendo o que veio. Se todas as cláusulas devolvem nil, a condicional OR devolve nil. Exemplo

```
>(OR t 5)
t
>(OR 5 t)
5
>(OR nil 5)
5
>(OR nil nil)
nil
```

Perceba-se que a razão para chamar AND e OR de condicionais e não de funções é que elas não avaliam imediatamente todos as suas cláusulas, eventualmente encerrando o ciclo ANTES de olhar todas. Isto tem conseqüências interessantes. Seja por exemplo uma função que deve elevar um número ao quadrado APENAS se for chamada com um número. Poderia ficar:

```
>(defun ELEVANUMERO (X)
(and (numberp X) (* X 2)))
>(ELEVANUMERO 3)
9
>(ELEVANUMERO 'P)
nil
```

Obviamente esta função poderia ser construída com if, senão vejamos:

```
>(defun ELEVA2 (X)
(if (numberp X) (* X 2) nil))
```

A mesma função, agora feita com COND

 ©88-09, Pedro Kantek

```
>(defun ELEVA3 (X)
(COND ((numberp X) (* X 2))
(t nil)))
```

A função NOT nega a cláusula. Assim:

```
>(NOT t)
nil
>(NOT nil)
t
>(NOT 5)
nil
```

As funções AND e OR, tais como as conhecemos em outras linguagens poderiam ser assim escritas:

```
>(defin FAND (A B)
(and A B t))
```

Esta função só admite 2 valores de entrada e apenas devolve t ou nil, nada mais.

EXERCÍCIO 11 Qual o valor das seguintes expressões?

```
(and (or (> 2 3) (not (= 2 3))) (< 2 3))
(not (or (= 1 2) (= 2 3)))
(or (< 1 2) (= 1 2) (> 1 2))
(and 1 2 3)
(or 1 2 3)
(and nil 2 3)
(or nil nil 3)

Resposta

T
T
T
3
1
nil
3
```

EXERCÍCIO 12 Escreva uma função que receba dois átomos extraídos de (AGULHA TESOURA PAPEL) e devolva 1 ou 2 dependendo de qual dos dois átomos ganhou o jogo do JOQUENPÔ.

EXERCÍCIO 13 Escreva um predicado que receba uma nota e uma freqüência e devolva t se o aluno estiver aprovado e nil se estiver reprovado. A aprovação é para nota maior ou igual a 7 e frequencia maior do que 75%

EXERCÍCIO 14 Escreva a função equivalente a (AND A1 A2 A3 A4) usando apenas IFs aninhados.

EXERCÍCIO 15 Escreva a função OR que funcione como OR de outras linguagens (t or t = t, t ot nil = t, nil or t = t, nil or nil = nil)

EXERCÍCIO RESOLVIDO 2 Escreva uma função que receba 1 átomo e uma lista e substitua o primeiro elemento da lista (2.operando) pelo primeiro operando Exemplo (EX1 'A '(B C D E F)) é (A C D E F)

```
> (defun ex1 (atomo lista)
   (cons atomo (cdr lista))
  )
EX1
> (ex1 'a '(1 2 3 4))
(A 2 3 4)
```

EXERCÍCIO RESOLVIDO 3 Escreva uma função que receba 2 átomos e uma lista e substitua todas as ocorrências do primeiro átomo na lista pelo segundo átomo. (EX2 'A 'B (A B C D E)) é (B B C D E)

```
> (defun ex2 (de por lista)   ; subst
   (cond
    ((null lista) nil)
    ((eql (car lista) de) (cons por (ex2 de por (cdr lista))))
    (t (cons (car lista) (ex2 de por (cdr lista))))
   )
  )
EX2
> (ex2 'a '1 '(b g a s e a 2 3))
(B G 1 S E 1 2 3)
> (ex2 'a 'b '(1 2 3 ))
(1 2 3)
```

Obs: esta função é a função primitiva *subst*

EXERCÍCIO RESOLVIDO 4 Escreva uma função que receba uma lista (com sublistas, se for o caso) e devolva a quantidade de átomos na lista

```
> (defun ex3 (lista)
   (cond
    ((null lista) 0)
    ((atom lista) 1)
    (t (+ (ex3 (car lista)) (ex3 (cdr lista))))
   )
  )
EX3
> (ex3 '(1 2 3 (4 5 6)))
6
> (ex3 '())
0

> (ex3 '(1 2 3 (4 5 6)))
6
> (ex3 '())
0
```

EXERCÍCIO RESOLVIDO 5 Escreva uma função EX13 que receba um número n e uma lista e devolva o n-ésimo elemento da lista

 ©88-09, Pedro Kantek

```
> (defun ex13 (n lista) ; nth
   (cond
    ((eql n 1) (car lista))
    (t (ex13 (- n 1) (cdr lista)))
   )
  )
EX13
> (ex13 3 '(a b c d e f ))
C
> (ex13 4 '(1 (1 2) (1 2 3) (1 2 3 4)))
(1 2 3 4)
> (ex13 6 '(1 2 3 4))
NIL
```

Esta função é a função primitiva *nth*.

EXERCÍCIO RESOLVIDO 6 Escreva uma função que receba um inteiro n, um elemento ele e uma lista lis e substitua o n-ésimo elemento da lista lis, pelo elemento ele. A origem da numeração é zero, ou seja o primeiro elemento da lista corresponde a n=0.

```
> (defun ex14 (n elemento lista)
   (cond
    ((= n 0) (cons elemento (cdr lista)))
    (t (cons (car lista) (ex14 (- n 1) elemento (cdr lista))))
   )
  )
EX14
> (ex14 3 'a '(1 2 3 4 5))
(1 2 3 A 5)
> (ex14 2 '(1 2) '(a (a b) (a b c) (a b c d)))
(A (A B) (1 2) (A B C D))
```

EXERCÍCIO RESOLVIDO 7 Escreva uma função que receba uma lista e devolva o número de elementos que ela tem

```
> (defun ex15 (lista) ;length
   (cond
    ((null lista) 0)
    (t (+ 1 (ex15 (cdr lista))))
   )
  )
EX15
> (ex15 '(1 2 3 4 5 6 7))
7
> (ex15 '(1 (1 2) (1 2 3)))
3
> (ex15 ())
0
```

EXERCÍCIO RESOLVIDO 8 Escreva uma função que receba um elemento e uma lista que contém esse elemento e devolva a posição do elemento na lista.

```
> (defun ex16 (elemento lista)
   (cond
```

```
    ((null lista) 0)
    ((eql elemento (car lista)) 1)
    (t (+ 1 (ex16 elemento (cdr lista))))
   )
  )
EX16
> (ex16 'gama '(alfa beta gama delta))
3
```

EXERCÍCIO RESOLVIDO 9 Escreva uma função que recebe duas listas e devolve outra lista obtida juntando as duas listas que entraram

```
> (defun ex17 (lista1 lista2) ; append
   (cond
    ((null lista1) lista2)
    (t (cons (car lista1) (ex17 (cdr lista1) lista2)))
   )
  )
EX17
> (ex17 '(1 2 3) '(a b c))
(1 2 3 A B C)
> (ex17 () '(a b c))
(A B C)
> (ex17 '(a b c) ())
(A B C)
```

Esta é a função primitiva *append*

EXERCÍCIO RESOLVIDO 10 Escreva uma função que receba uma lista e devolva-a com seus elementos em ordem invertida

```
> (defun ex18 (lista) ; reverse
   (labels
    ((ex18-aux (lista lista-aux)
     (cond
      ((null lista) lista-aux)
      (t (ex18-aux (cdr lista) (cons (car lista) lista-aux)))
     )
    ))
   (ex18a-aux lista nil))
  )
EX18
> (ex18 '(a b c d e (f g)))
((F G) E D C B A)
```

Esta função existe em Lisp e se chama *reverse*. Outra observação que deve ser feita aqui é o uso da macro *labels* que permite definir sub-funções dentro de uma função. A vantagem aqui é que a sub-função tem acesso às variáveis locais da função principal. Note que no caso acima, primeiro define-se a função ex18-aux, e depois chama-se-á na última linha. A desvantagem é que o trace nestes casos não funcional legal.

EXERCÍCIO RESOLVIDO 11 Escreva uma função que recebe uma lista (eventualmente contendo sublistas) e devolve a mesma lista original só que em ordem reversa. As sublistas porventura existentes também devem vir em ordem reversa.

 ©88-09, Pedro Kantek

```
> (defun ex19 (lista)
   (labels
    ((ex19-aux (lista lista-aux)
     (cond
      ((null lista) lista-aux)
      ((atom lista) lista)
      (t (ex19-aux (cdr lista) (cons (ex19 (car lista)) lista-aux)))
     )
    ))
    (ex19-aux lista nil))
  )
EX19
> (ex19 '(a b ((1 2 3)) (x y z) 8))
(8 (Z Y X) (3 2 1) B A)
> (ex19 '(a b ((1 2 3)) (x y z) 8))
(8 (Z Y X) ((3 2 1)) B A)
```

EXERCÍCIO RESOLVIDO 12 Escreva uma função que receba uma função e uma lista e devolva outra lista que é o resultado da função aplicada a cada um dos elementos da lista original.

```
> (defun ex20 (funcao lista) ; mapcar
   (cond
    ((null lista) nil)
    (t (cons (funcall funcao (car lista)) (ex20 funcao (cdr lista))))
   )
  )
EX20
> (defun dobro (n) (* 2 n))
DOBRO
> (ex20 'dobro '(1 2 3 4))
(2 4 6 8)
>(mapcar 'dobro '(2 3 4))
(4 6 8)
```

Esta função é a primitiva *mapcar*.

EXERCÍCIO RESOLVIDO 13 Escreva uma função que receba uma lista eventualmente contendo sublistas e devolva uma lista que tenha os mesmos elementos, mas que não contenha sublistas.

```
> (defun ex21 (lista)
   (cond
    ((null lista) nil)
    ((atom lista) (list lista))
    (t (append (ex21 (car lista)) (ex21 (cdr lista))))
   )
  )
EX21
> (ex21 '(1 2 (3 4 (5 6 )) 7))
(1 2 3 4 5 6 7)
> (ex21 '((a b) c d ((f g h)) i))
(A B C D F G H I)
```

EXERCÍCIO RESOLVIDO 14 Escreva uma função que receba um elemento e uma lista e responda se o elemento pertence ou não a lista.

```
> (defun ex22 (elemento lista) ; member
   (cond
    ((null lista) nil)
    ((eql elemento (car lista)) t)
    (t (ex22 elemento (cdr lista)))
   )
  )
EX22
> (ex22 'a '(2 2 3 4 5))
NIL
> (ex22 'a '(1 2 3 a b c))
T
```

Esta função é a primitiva *member*. Quando ela encontra um elemento igual na lista, devolve o resto dessa lista

EXERCÍCIO RESOLVIDO 15 Escreva uma função que receba um elemento e uma lista e devolva outra lista onde esse elemento não aparece.

```
> (defun ex23 (elemento lista) ; remove
   (cond
    ((null lista) nil)
    ((eql elemento (car lista)) (ex23 elemento (cdr lista)))
    (t (cons (car lista) (ex23 elemento (cdr lista))))
   )
  )
EX23
> (ex23 '1 '(4 3 2 1 1 2 3 4))
(4 3 2 2 3 4)
> (ex23 'a '(1 2 3))
(1 2 3)
> (ex23 '1 '(1 1 1))
NIL
```

A primitiva em lisp chama-se *remove*

EXERCÍCIO RESOLVIDO 16 Escreva uma função que analise uma lista e remova os elementos duplicados dela.

```
> (defun ex24 (lista)
   (cond
    ((null lista) nil)
    ((member (car lista) (cdr lista)) (ex24 (cdr lista)))
    (t (cons (car lista) (ex24 (cdr lista))))
   )
  )
EX24
> (ex24 '(1 2 3 4 4 3 2 1))
(4 3 2 1)
```

Note que esta função não preserva a ordem da lista original. Para preservá-la tem-se que olhar a lista que se está construindo e não a que se está analisando. Veja aqui:

 ©88-09, Pedro Kantek

```
> (defun ex24a (lista)  ; remove-duplicates
   (labels
    ((ex24a-aux (lista lista-aux)
     (cond
      ((null lista) nil)
      ((member (car lista) lista-aux) (ex24a-aux (cdr lista) lista-aux))
      (t (cons (car lista) (ex24a-aux (cdr lista) (cons (car lista) lista-aux))))
     )))
     (ex24a-aux lista nil)
   )
  )
EX24A
> (ex24a '(1 2 3 4 4 3 2 1))
(1 2 3 4)
```

Esta função em lisp chama-se *remove-duplicates*

EXERCÍCIO RESOLVIDO 17 Escreva função que faça o contrário do cons, a saber: ela deve receber um elemento e uma lista e juntar o elemento no final da lista

```
> (cons 'a '(1 2 3 4))
(A 1 2 3 4)
> (defun ex25 (elemento lista)
   (cond
    ((null lista) (list elemento))
    (t (cons (car lista) (ex25 elemento (cdr lista))))
   )
  )
EX25
> (ex25 'a '(1 2 3 4))
(1 2 3 4 A)
```

EXERCÍCIO RESOLVIDO 18 Escreva uma função que faça o contrário da car: devolver o último elemento da lista.

```
> (defun ex26 (lista)
   (cond
    ((null (cdr lista)) (car lista))
    (t (ex26 (cdr lista)))
   )
  )
EX26
> (car '(1 2 3 4))
1
> (ex26 '(1 2 3 4))
4
```

EXERCÍCIO RESOLVIDO 19 Escreva uma função que faça o contrário de cdr: devolver toda a lista menos o último elemento.

```
> (defun ex27 (lista)
   (cond
    ((null (cdr lista)) nil)
    (t (cons (car lista) (ex27 (cdr lista))))
```

```
   )
  )
EX27
> (cdr '(1 2 3 4))
(2 3 4)
> (ex27 '(1 2 3 4))
(1 2 3)
```

Esta função existe em lisp e se chama *butlast*.

EXERCÍCIO RESOLVIDO 20 Escreva uma função que receba uma função e uma lista e verifique se a função é verdade para todos os elementos da lista.

```
> (defun ex28 (funcao lista) ; every
   (cond
    ((null lista) t)
    ((funcall funcao (car lista)) (ex28 funcao (cdr lista)))
    (t nil)
   )
  )
EX28
> (ex28 'atom '(1 2 3 4))
T
> (ex28 'atom '(1 2 (3 4)))
NIL
```

Esta função se chama *every*

EXERCÍCIO RESOLVIDO 21 Escreva uma função que receba uma função e uma lista e verifique se a função é verdade para pelo menos um dos elementos da lista.

```
> (defun ex29 (funcao lista) ; some
   (cond
    ((null lista) nil)
    ((funcall funcao (car lista)) t)
    (t (ex29 funcao (cdr lista)))
   )
  )
EX29
> (ex29 'atom '(1 2 (3 4)))
T
> (ex29 'atom '((1 2) (3 4) ((5))))
NIL
```

O nome em lisp é *some*

EXERCÍCIO RESOLVIDO 22 Escreva uma função que receba uma função e uma lista e verifique se a função NÃO é verdade para todos os elementos da lista.

```
> (defun ex30 (funcao lista) ; notany
   (cond
    ((null lista) t)
    ((not (funcall funcao (car lista))) (ex30 funcao (cdr lista)))
    (t nil)
```

 ©88-09, Pedro Kantek

```
   )
  )
EX30
> (ex30 'atom '((1 2) (3 4)))
T
```

Chama-se *notany*

EXERCÍCIO RESOLVIDO 23 Escreva uma função que receba uma função e uma lista e verifique se a função NÃO é verdade para pelo menos um dos elementos da lista.

```
> (defun ex31 (funcao lista)
   (cond
    ((null lista) nil)
    ((not (funcall funcao (car lista))) t)
    (t (ex31 funcao (cdr lista)))
   )
  )
EX31
> (ex31 'atom '(1 2 (3 4)))
T
```

O nome em lisp é *notevery*.

Quatro erros na definição de função

Seja a seguinte função LISP correta

```
(defun COMER (A B)
(list 'GOSTO 'DE 'COMER A 'COM B))

Assim, a chamada de
(COMER LARANJA MACARRÃO) gera (GOSTO DE COMER LARANJA COM MACARRÃO)
```

Erro 1: Escrever parênteses ou aspas na lista de argumentos. Exemplo:

```
(defun COMER ('A 'B)
(list 'GOSTO 'DE 'COMER A 'COM B))
ou
(defun COMER ((A) (B))
(list 'GOSTO 'DE 'COMER A 'COM B))
```

Erro 2: Colocar parênteses nas variáveis dentro do corpo da função. Exemplo:

```
(defun COMER (A B)
(list 'GOSTO 'DE 'COMER (A) 'COM (B)))
```

Erro 3: Colocar aspas nas variáveis

```
(defun COMER (A B)
(list 'GOSTO 'DE 'COMER 'A 'COM 'B))
Assim, a chamada de
(COMER LARANJA MACARRÃO) gera (GOSTO DE COMER A COM B)
```

Erro 4: Não colocar aspas no que não é variável e sim conteúdo

```
(defun COMER (A B)
(list GOSTO DE COMER A COM B))
Assim, a chamada de
(COMER LARANJA MACARRÃO) gera um erro: "função GOSTO desconhecida...")
```

7.3.1 Função APPLY

A função APPLY recebe como parâmetros um nome de função e um conjunto de parâmetros e aplica (apply) esta função aqueles parâmetros. Veja-se no exemplo:

```
> (apply 'CONS 'A 'B))
(A.B)
```

7.3.2 Variáveis globais e locais

Quando se define uma variável fora do cabeçalho de uma função, ela é considerada global e vale para todas as funções e fórmulas emitidas a partir daquele ponto. Uma variável global sempre é obtida a partir da função SET ou da função SETQ.

As variáveis locais, são aquelas escritas nos cabeçalhos das funções definidas pelo usuário. Por exemplo, em

```
>(defun SOMA (A B)
  (+ A B))
```

As variáveis A e B são locais e tem existência apenas enquanto durar a execução da função SOMA. Quando esta função é chamada, por exemplo, com 2 e 3, como em:

```
>(SOMA 2 3)
5
```

O que o LISP faz é assinalar A para 2 e B para 3 e então resolver a função SOMA. Quando SOMA termina A e B desaparecem.

Existem duas funções BOUNDP e MAKUNBOUND que manuseiam variáveis globais. BOUNDP nome devolve t se existir uma variável (global) com nome enquanto MAKUNBOUND nome faz desaparecer a variável (global) nome.

EQ e EQUAL

EQUAL é um predicado que retorna t se seus operandos forem iguais e nil senão. EQ é um predicado mais restrito do que EQUAL e que só retorna t se seus operandos forem os mesmos (e não apenas iguais). Para poder entender esta diferença há que se lembrar como o LISP armazena variáveis e acompanhar o exemplo

```
>(EQUAL 2 2)
t
>(EQ 2 2)
t
>(EQUAL '(A B) '(A B))
t
>(EQ '(A B) '(A B))
nil
```

Em resumo, a igualdade numérica é estabelecida por "=". Dois símbolos são eq se e apenas se eles são idênticos. Duas cópias da mesma lista não são eq, mas são equal.

```
> (eq 'a 'a)
T
> (eq 'a 'b)
NIL
> (= 3 4)
T
> (eq '(a b c) '(a b c))
NIL
```

 ©88-09, Pedro Kantek

```
> (equal '(a b c) '(a b c))
T
> (eql 'a 'a)
T
> (eql 3 3)
T
```

O predicado eql é equivalente a eq para símbolos e a = para números. O predicado equal é equivalente a eql para símbolos e números. Ele é verdadeiro para duas conses se e apenas se os CAR's e os CDR's dessas conses são equal. Ele é verdadeiro para duas estruturas se e somente se as estruturas são do mesmo tipo e seus correspondentes campos são equal.

Quando se define uma função em LISP e se definem para ela dois argumentos, digamos X e Y, estabelece-se que na chamada da função, dois argumentos deverão ser passados. Através de um fenômeno conhecido em LISP como "binding", há uma associação entre X e o primeiro argumento passado e entre Y e o segundo. Essa ligação ocorre apenas durante a execução da função.

Podem-se ser definidos parâmetros opcionais para as funções. Para isso se usa a palavra chave "&optional". Qualquer coisa definida depois desta palavra, é opcional. Veja os exemplos

```
>(defun AA (X &optional Y) (if Y X 0))
AA
>(defun BB (&optional (X 3) (Z 10)) (+ X Z))
BB
>(AA 5)
0
>(AA 5 t)
5
>(BB 5)
15
>(BB 5 6)
11
>(BB)
13
```

7.3.3 Impressão

Algumas funções imprimem resultados. A mais simples é a função "print" que imprime seu argumento e depois retorna-o.

```
Veja o exemplo
>(print 11)
11    ; este é o resultado da impressão
11    ; este é o retorno da função print
```

Se uma impressão mais sofisticada é necessária, deve-se usar a função "format". Veja o exemplo

```
>(format t "Um atomo: ~S~%e uma lista: ~S~%e
   um inteiro: ~D~%" nil (list 5) 7)
```

Um atomo: NIL e uma lista: (5) e um inteiro: 7

O primeiro argumento de format é t, nil ou um string. T especifica saida no terminal. Nil significa que nenhuma saida deve se dar, mas que deve ser retornada saida formatada.

Um string especifica um lugar genérico para se dar a saída. Pode ser um arquivo, o terminal ou outro programa.

O segundo argumento é a máscara de formatação. Ele é um string contendo diretivas de formatação. Neste caso, LISP obedece a tais diretivas para montar a saída. Format sempre retorna nil, a menos que o primeiro argumento seja nil.

Alguns valores permitidos como diretivas são:
`~S`: esta diretiva imprime qualquer objeto LISP tal como ele seria impresso pelo interpretador
`~D`: imprime apenas valores inteiros
`~%`: imprime um CR+LF (pula a linha)
`~~`: substituído por um único

Exemplos: ... ...

7.3.4 Função LET

Esta função permite estabelecer valores para variáveis dentro de blocos de código. Seu formato é

```
>(let ((variável1 valor1)
       (variável2 valor2)
       ...
           )
       execução
 )
```

Este comando assinala os valores citados a cada variável colocada antes do valor e depois executa o trecho de código estabelecido em "execução". Acompanhe os exemplos

```
> (let ((a 3)) (+ a 1))
4
> (let ((a 2)
        (b 3)
        (c 0))
    (setq c (+ a b))
    c
  )
5
> (setq c 4)
4
> (let ((c 5)) c)
5
> c
4
```

Vejamos porque é importante a presença do LET. Seja a função

```
> (defun moeda ()
    (cond
       ((< (random 101) 50) 'cara)
       ((> (random 101) 50) 'coroa)
       ((equal (random 101) 50) 'em-pé)))
```

Esta função contém um erro, senão vejamos: Suponha que ao executá-la a primeira chamada de random entregue 80. Como a primeira condição é falsa, passa-se à seguinte:

 ©88-09, Pedro Kantek

a segunda chamada de random entrega 30. Como a segunda condição também é falsa, passa-se a terceira chamada de random que entrega 90. Como 90 é diferente de 50, a função devolve nil, que passaria a ser um "quarto estado da moeda".

Para corrigir este defeito, eis a função correta:

```
> (defun moeda ()
    (let ((insta (random 101)))
    (cond
       ((< insta 50) 'cara)
       ((> insta 50) 'coroa)
       (t 'em-pé)))
```

7.3.5 Interação

A interação mais simples em LISP é o LOOP. Seu formato é

```
(LOOP
 (alguma coisa
 (when (predicado) (return variável))
)
```

Acompanhe nos exemplos:

```
> (setq a 4)
4
> (loop
   (setq a (+ a 1))
   (when (> a 7) (return a))
  )
8
> (loop
   (setq a (- a 1))
   (when (< a 3) (return))
  )
NIL
```

A próxima função é DOLIST: ela liga uma variável aos elementos de uma lista e pára quando o fim da lista é encontrado.

```
> (dolist (x '(a b c)) (print x))
A
B
C
NIL
```

DOLIST sempre retorna nil. O valor de x no exemplo acima nunca é NIL.

DO O último é o DO. Veja seu formato

```
(DO ((variável-1 valor-inicial1 [atualização-1])
     (variável-2 valor-inicial2 [atualização-2]) ...)
    (teste ação-1 ... ação-n)
     corpo )
```

Por outro lado PUSH e POP são maneiras elegantes de acrescentar e escluir elementos de listas. Acompanhe nos exemplos:

```
> (setf lista nil)
> (push 'azul lista)  ; a lista agora é (azul)
> (push 'verde lista) ; a lista agora é (verde azul)
> (push 'branco lista); a lista é (branco verde azul)
> (pop lista)
branco ; a lista agora é (verde azul)
```

Eis as definições de push e pop

```
> (push atomo lista) ; equivale a
> (setf lista (cons atomo lista))
>
> (pop lista) ; equivale a
> (let ((x (car lista)))
   (setf lista (cdr lista))
   x)
```

```
> (do ((x 1 (+ x 1))
       (y 1 (* y 2)))
      ((> x 5) y)
    (print y)
    (print 'working)
  )
1
WORKING
2
WORKING
4
WORKING
8
WORKING
16
WORKING
32
```

A primeira parte de um DO especifica quais variáveis ligar e com quais valores e como elas devem ser atualizadas. A segunda parte especifica uma condição de terminação e o valor que deve ser retornado. A última parte é o que deve ser executado. Na execução DO assinala as variáveis e verifica a condição de fim. Se a condição é falsa, o corpo da função é executado repetidamente. Quando a condição se torna verdadeira, DO retorna o valor especificado.

Eis como ficaria o formato

```
(do ((variável velor-inicial atualização)...)
     ((quando parar) (o que fazer quando parar))
    ((corpo da iteração)...)
)
Assim, no exemplo acima,
> (do
((x 1 (+ x 1))(y 1 (* y 2)))  ; variáveis inicializadas
                              ; com seus incrementos
((> x 5) y)                   ; quando parar e o que fazer
                              ; quando parar
 (print y) (print 'working)   ; corpo da repetição
)                             ; final do comando
```

 ©88-09, Pedro Kantek

Acompanhe-se aqui, duas implementações para o mesmo problema nas versões recursiva e iterativa:

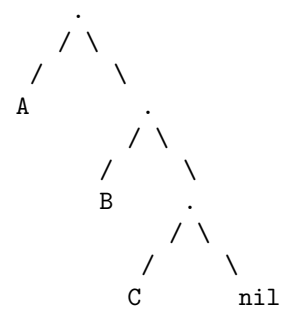
```
> ; versão recursiva
> (defun quadrados (quem fim)
    (if (> quem fim)
      'feito
      (progn
        (format t "~A ~A~%" quem (* quem quem))
        (quadrados (+ quem 1) fim))))

> ; versão iterativa
> (defun quadrados (inicio fim)
    (do (i inicio (+ i 1)))
       ((> i fim) 'feito)
       (format t "~A ~A~%" quem (* quem quem))))
```

7.4 Árvores: Representação com ponto

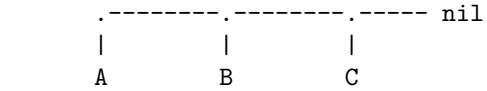
Em alguns casos particulares uma célula LISP contém no seu segundo endereço algo que não é um apontador. Nestes casos, o LISP imagina que isto não é uma lista e ao representar este conteúdo ele é obrigado a usar uma representação diferente na qual o primeiro e o segundo conteúdo da célula estão separados por um ponto.

Vejamos: Seja o conse contendo o nome de uma pessoa (JOAO) e a sua idade (34). Dentro do LISP isto é representado como (JOAO . 34). O elemento à esquerda de uma conse é chamado de car da conse. O elemento a direita é chamado de cdr da conse. Tanto o car quanto o cdr de uma conse, podem ser conses também.

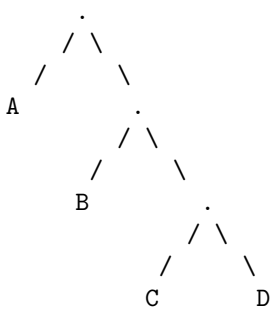
Por exemplo, o nome do JOAO poderia ser JOAO.SILVA. A conse completa ficaria: ((JOAO.SILVA).34). LISP representa uma lista como uma coleção de conses ligadas, na qual o car de cada conse aponta para o elemento da lista, e o cdr de cada conse aponta para o próximo conse. O ultimo elemento aponta para NIL. Seja por exemplo a lista (A B C). No LISP ela é como:



Note-se que se fizermos uma rotação de 45 graus para a esquerda, a lista fica bem mais maneira:



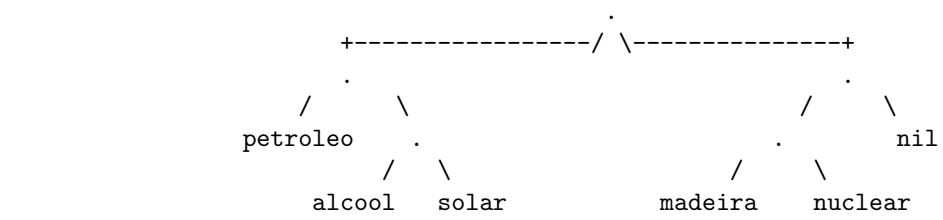
Esta estrutura linear sugere que é muito mais fácil ler e representar as listas no formato (A B C) do que no (A . (B . (C . nil))), que entretanto são absolutamente equivalentes. O LISP usa a notação linear quando possível e a notação de ponto quando necessário. Por exemplo, a estrutura a seguir, que não termina por NIL,



é representada como (A B C . D)

De qualquer maneira o leitor do LISP aceita os dois tipos de notação, inclusive misturados. CAR e CDR quando aplicados a arvores extraem respectivamente o filho esquerdo e direito da arvore. Vejamos um exemplo

```
> (setq energia '((petroleo . (alcool.solar)).(madeira.nuclear)))
Esta estrutura pode ser assim desenhada:
```




```
> (car energia)
   (petroleo . (alcool . solar))
> (cdr energia)
   (madeira . nuclear)
> (car (car energia))
   petroleo .
> (cdr (car energia))
   ((alcool . solar))
> (car (cdr (car energia)))
   (alcool . solar)
> (car (car (cdr (car energia))))
   alcool
```

Até quatro chamadas de CAR e CDR podem ser compostas: (existem 28 composições CAAAAR = car do car do car do car CADADR = car do cdr do car do cdr CAAR = car do car CDDAR = cdr do cdr do car ...

```
Exemplo > (setq nu '(1 2 3 4 5)) (1 2 3 4 5)
> nu (1 2 3 4 5)
> (caddr nu)     => car do cdr do cdr de nu                  3
> (cadddr nu)    => car do cdr do cdr do cdr de nu           4
> (caddddr nu)   => mais de 4                          ... ERRO...
> (cadadr nu)    => car do cdr do car de cdr de nu           nil
> (cdddr nu)     => cdr do cdr do cdr de nu                  (4 5)
```

Seja agora fazer uma função chamada ULTIMO que devolve o último elemento de uma lista

 ©88-09, Pedro Kantek

```
> (defun ul (li)
    (cond
      ((null li) nil)
      ((eql nil (cdr li)) (car li))
      (t (ul (cdr li)))))
UL
> (ul '(1 2 3 4))
4
> (ul '((a b c)(1 2)(3 k j l)))
(3 K J L)
```

Outro exemplo usando LAST

```
> (last '(23 54 -23 15 . 27))
 antes de responder, vamos construir a árvore expressa:
                    .
                  /   \
                23      .
                      /   \
                    54      .
                          /   \
                       -23     .
                             /   \
                           15    27
```

Vista a arvore (note que numa construção tipo conse, na ponta só pode ter um átomo ou outra conse, nunca uma lista). a resposta só pode ser (15.27) Existe uma função primitiva de extração chamada NTH, para extrair o n-esimo elemento de uma lista. Sendo n um inteiro não negativo (a base de contagem é sempre zero), nth n trará o dito cujo. Se ele não existir deve voltar NIL. Seja agora definir uma função INDEXER tal como descrito no exemplo

```
> (indexer '(A B C D E F G) '(6 2 4 0))
(G C E A)
```

pode ser assim:

```
> (defun indexer (LST1 LST2)
    (cond
      ((null LST2) nil)
      (t (cons (nth (car LST2) LST1) (indexer lst1 (cdr LST2))))))
INDEXER
> (indexer '(a b c d e f g) '(6 2 4 0))
(G C E A)
```

Existe também a função NTHCDR que devolve o n-esimo CDR da lista operando.

```
> (nthcdr 0 '(BOOK PENCIL PAPER PEN))
(BOOK PENCIL PAPER PEN)
> (nthcdr 2 '(BOOK PENCIL PAPER PEN))
(PAPER PEN)
> (nthcdr 5 '(BOOK PENCIL PAPER PEN))
nil
```

Note que tanto nth como nthcdr devolvem nil se não houver elementos suficientes na lista

EXERCÍCIO 16 Tente responder antes do LISP (cubra a resposta com uma folha de papel...)

1. Montar a seguinte árvore
(((IBM . PC) . (APPLE . MACINTOSH)) . (TANDY . TRS-80))
Uma resposta pode ser
> (cons(cons(cons 'ibm pc)(cons 'apple mac))(cons 'tandy trs))
(((IBM . PC) APPLE . MACINTOSH) TANDY . TRS-80)
Outra:
> (cons (CONS (CONS IBM PC) (CONS APPLE MACINTOSH))
(CONS TANDY TRS-80))
(((IBM . PC) APPLE . MACINTOSH) TANDY . TRS-80)
Note que parênteses desnecessários NÃO são escritos.

7.5 Operadores aplicativos

É uma função que recebe outra função como operando e aplica-a (daí o nome) aos operandos entregues.

Eis alguns exemplos:

```
> (funcall '+ 3 4)
7
> (apply '+ 3 4 '(3 4))
14
> (mapcar 'not '(t nil t nil t nil))
(NIL T NIL T NIL T)
```

FUNCALL FUNCALL aplica seu primeiro argumento aos demais. Por exemplo:

```
> (funcall 'cons 'a 'b)
(a . b)
```

Outro exemplo, ilustrativo:

```
> (setf fn 'cons) ; note que setf é uma generalização de setq
> (type-of fn)
COMPILED-FUNCTION
> (funcall fn 'c 'd)
(c . d)
```

Usando FUNCALL, pode-se escrever um operador próprio que por sua vez toma uma função como entrada

APPLY APPLY é como funcall, exceto que o último argumento deve ser uma lista. Os elementos dessa lista são tratados como argumentos para funcall.

MAPCAR É o operador aplicativo mais freqüentemente usado. Ele aplica uma função a cada um dos elementos de uma lista, e retorna outra lista contendo os resultados obtidos. O primeiro argumento de mapcar deve ser uma função de um argumento. Acompanhe no exemplo:

 ©88-09, Pedro Kantek

```
> (defun dobro (n)
    (* 2 n))
> (dobro 6)
12
> (dobro '(1 2 3 4 5))
ERRO: tipo inválido
> (mapcar 'dobro '(1 2 3 4 5)
(2 4 6 8 10)
```

A quantidade de elementos da lista de saída é igual à quantidade de elementos na lista de entrada. Assim, se chamado com uma lista vazia, a resposta é vazia também:

```
> (mapcar 'dobro '() ); poderia ser também (mapcar ' ())
nil
```

Seja um exemplo de manipulação de tabelas:

```
> (setf dic '((um un) (dois deux) (tres trois)
   (quatro quatre) (cinco cinq)))
```

Contagem em português:

```
> (defun primeiro (x)
    (car x))
> (defun segundo (x)
    (car (cdr x)))
> (mapcar 'primeiro dic)
(UM DOIS TRES QUATRO CINCO)
> (mapcar 'segundo dic)
(UN DEUX TROIS QUATRE CINQ)
```

Invertendo o dicionário

```
> (mapcar 'reverse dic)
((UN UM) (DEUX DOIS) (TROIS TRES) (QUATRE QUATRO) (CINQ CINCO))
```

7.5.1 Notação LAMBDA

Foi inventada pelo lógico americano Alonzo Church na Universidade de Princeton, na década de 30. O seu objetivo era escrever um método não ambíguo de especificar funções informando suas entradas e os cálculos que deveriam realizar. A função lambda que divide um número por 2 é representada por λx.(x/2)

Há duas maneiras de representar uma função para ser usada em um operador aplicativo. A primeira, é a que se viu acima. Define-se a função através de `defun`, e depois ela é chamada usando o sinal '.

A segunda maneira é passar a função diretamente através da macro lambda. O resultado da utilização de `lambda` é uma função, assim a mesma pode ser passada para o operador mapcar. Com isso, se evita ter que definir a função em duas etapas. Veja-se o exemplo:

```
> (mapcar '(lambda (n) (* 2 n)) '(1 2 3 4 5))
(2 4 6 8 10)
```

Em outras palavras, se você precisa criar uma função temporária, e não quer pensar num nome para ela, você deve usar a função LAMBDA .

A combinação de lambda e mapcar pode ser usada em lugar de loops. Por exemplo, as seguintes duas formas são equivalentes:

```
> (do ((x '(1 2 3 4 5) (cdr x))
       (y nil))
      ((null x) (reverse y))
    (push (+ (car x) 2) y)
  )
(3 4 5 6 7)
> (mapcar '(lambda (x) (+ x 2)) '(1 2 3 4 5))
(3 4 5 6 7)
```

operador FIND-IF Este operador recebe um predicado e uma lista e devolve o primeiro elemento da lista para o qual o predicado retorna não nil (Lembre que o elemento retornado é o próprio pesquisado). Se nenhum elemento satisfaz o predicado, o operador retorna nil. Veja-se o exemplo

```
> (find-if 'numberp '(a b c 2 3 6))
2
> (find-if '(lambda (x) (> x 10)) '(1 11 21 31 41))
11
```

Seja agora um exemplo um pouco mais complexo. Escreva uma função que receba um átomo e uma lista e devolva o primeiro elemento da lista que seja diferente do que o átomo e menor que o dobro do atomo

```
> (defun exercicio (atomo lista)
    (find-if '(lambda (x) (and (not (equal atomo x))
                           (> (* atomo 2) x))) lista))
(exercicio 100 '(500 100 102 300 180))
102
```

operador REMOVE-IF Este operador recebe um predicado e uma lista, e remove todos os elementos da lista que satisfazem ao predicado, devolvendo-a sem eles. Veja os exemplos

```
> (remove-if 'numberp '(1 2 3 de oliveira 4))
(DE OLIVEIRA)
> (remove-if 'oddp '(1 2 3 4 5 6))
(2 4 6)
```

Outro operador bastante usado é REMOVE-IF-NOT, que funcional tal como REMOVE-IF, só que invertendo o sentido do predicado. Aqui são removidos os elementos que não satisfazem ao predicado.

REDUCE O operador reduce reduz os elementos da lista a um único resultado. É similar ao operador REDUCTION do APL (lá é uma /). Aqui, a função passada como parâmetro pode ser qualquer *mas desde que aceite dois operandos*. Veja o exemplo:

```
> (reduce '+ '(1 2 3 4 5))
15
> (reduce '* '(1 2 3 4 5))
120
```

Um uso interessante para reduce é obter uma lista simples quando há uma lista de listas. Usa-se para isso a função append. Veja no exemplo:

```
> (reduce 'append '((ana maria) (jose eduardo)
            (paula roberta) (felipe antonio)))
(ANA MARIA JOSE EDUARDO PAULA ROBERTA FELIPE ANTONIO)
```

 ©88-09, Pedro Kantek

7.5.2 Classificação

LISP provê duas primitivas para ordenar: SORT e STABLE-SORT

```
> (sort '(2 1 5 4 6) '<)
(1 2 4 5 6)
> (sort '(2 1 5 4 6) '>)
(6 5 4 2 1)
```

O primeiro argumento de um SORT é uma lista, o segundo é uma função de comparação. Ambos tem a mesma função, mas SORT não garante estabilidade: Se há dois elementos a e b tais que (and (not (< a b)) (not (< b a))) então a função sort pode devolvê-los em quaisquer ordem. stable-sort garante que se a condição acima se verificar, a ordem de devolução será exatamente a ordem de entrada. Ambas as funções destroem seus argumentos, assim se eles serão necessários depois é conveniente passar apenas uma cópia deles.

7.5.3 Algumas funções interessantes

```
> (append '(1 2 3) '(4 5 6))    ;concatena listas
(1 2 3 4 5 6)

> (reverse '(1 2 3))            ;reverte os elementos de uma lista
(3 2 1)

> (member 'a '(b d a c))      ;Entrega a sublista que começa com elem.
                    ;Se ele não estiver em lista devolve NIL
(A C)

> (find 'a '(b d a c))            ;encontra o primeiro elemento na lista
                                       ;que começa pelo operando
A

> (find '(a b) '((a d) (a d e) (a b d e) ())
(A B D E)

> (subsetp '(a b) '(a d e))    ;o primeiro é um subconjunto do segundo ?
NIL

> (intersection '(a b c) '(b))  ; conjunto intersecção
(B)

> (union '(a) '(b))            ; conjunto união
(A B)

> (set-difference '(a b) '(a))  ; conjunto diferença
(B)
```

7.5.4 Entrada e Saída em LISP

As definições a seguir valem apenas para o COMMON LISP, uma vez que estas definições são fortemente dependentes do ambiente.

Uma primeira questão é como efetuar uma entrada de dados para uma função que os necessite. Acompanhe no exemplo:

```
> (defun pergunte (string)
    (format t "~A" string)
    (read))
PERGUNTE
> (pergunte "Qual a cotação do dólar ?")
Qual a cotação do dólar ?2.96
2.96
> (setq vari (pergunte "Quem é você ?)
Quem é você ?Pedro
> vari
PEDRO
```

Arquivos

Abertura e leitura de arquivos A diretiva em LISP é open, que ao abrir um arquivo cria uma variável (descritor) do mesmo que passa a ser referenciada dentro do LISP. Assim, deve-se fazer:

```
(setq nome-variável (open "caminho e nome-do-arquivo"))
```

ou, se o nome é simples (não contem barras ou dois pontos...)

```
(setq nome-variável (open 'arquivo))
```

Se este comando for bem sucedido a variável "nome-variável"será criada e comandará as operações de entrada e saída. Agora, a cada vez que a função

```
(setq armazenagem (read nome-variável))
```

for executada, em armazenagem existirá uma palavra do arquivo originalmente aberto. Variações sobre este tema

```
> (setq alfa (read arquivo nil 'eof))
```

Retorna palavra a palavra do arquivo "arquivo"até que ele termine. A partir daí ele passa a retornar EOF

```
> (setq alfa (read-line arquivo nil 'eof))
```

Retorna linha a linha do arquivo. As linhas vêm entre aspas duplas. Ao se encerrar o arquivo vem a literal EOF.

As duas implementações acima foram testadas em CLISP e em Lispworks e ambas funcionaram bem.

Após aleitura de arquivo , o mesmo deve ser fechado com a função

```
(close nome-variável)
```

Gravação de dados Para gravar informações, o ciclo é o mesmo, mas a operação de "open"deve ter algumas informações a mais:

```
(setq variável (open 'arquivo :direction :output))
```

A partir deste ponto, a gravação pode ser fazer via função write (veja o cltl2) ou mais facilmente usando a família de comandos PRIN. Veja o exemplo, supondo que o arquivo de saída foi aberto gerando a variável ARQ1 (setq ARQ1 (open 'blablabla :direction :output))...

 ©88-09, Pedro Kantek

```
(PRIN1 objeto-LISP-qualquer ARQ1)
      ;; note a presença do paramentro adicional ARQ1
```

Neste caso, o fechamento do arquivo passa a ser mandatório (ao contrário dos arquivos abertos para leitura) e deve ser comandado via

```
(close ARQ1)
```

Observação importante: Os nomes de arquivos podem ser os nomes do sistema operacional (LPT1, por exemplo). Neste caso, a saída se dá diretamente na impressora. Eis o exemplo completo:

```
(setq a (open 'LPT1 :direction :output))
(prin1 objeto a)
(close a)
```

PROG = significa programa. (PROG (variáveis-prog) forma1 forma2...) variáveis-prog são variáveis locais que recebem nil no início de prog. qualquer uma delas pode ser mudada por SETQ . As formas a seguir vão sendo avaliadas normalmente. Ao final PROG devolve nil. PROG1, PROG2 e PROGN são iguais a PROG, só que devolvem o resultado da primeira, segunda e última expressões.

TERPRI = move o cursor a uma nova linha

PRINC e PRIN1 = imprimem objetos LISP

Leitura e gravação em COMMON LISP 1. Leitura: `(setq arq (open 'nome-completo-do-arquivo))`
depois disso fazer: `(setq variavel (read arq))`
e os dados do arquivo estarão na variável.

2. Gravação

```
(setq arq (open 'nome-completo :direction :output))
depois
(prin1 objeto-a-gravar arq)
ao final
(close arq)
```

Note que 'nome-completo pode ser LPTx, e daí os dados saem direto.

EXERCÍCIO 17 Escreva uma função LISP chamada CENSURA que receba dois operandos: O primeiro é uma lista de palavras censuradas e o segundo é o nome de um arquivo que contém um texto. A função deve ler o arquivo e comparar palavra a palavra do mesmo com a lista de palavras a censurar. Quando houver a igualdade, a palavra original deve ser substituída pela palavra CENSURADO. A saída deve ser gerada na própia tela. Se o usuário quiser gravar, ele que providencie.

```
> (defun censura (lpc arq)
    (COND
      ((null arq) nil)
      ((member (car arq) lpc) (cons 'censurado (censura lpc (cdr arq))))
      (t (cons (car arq) (censura lpc (cdr arq))))))
CENSURA
> (censura '(coxa branca) '(a grande nacao coxa e a sua
equivalente branca sairam))
(A GRANDE NACAO CENSURADO E A SUA EQUIVALENTE CENSURADO SAIRAM)
```

7.5.5 Para você fazer

Descubra o que fazem estas funções lisp

```
1. (defun AAA (x)
  (and (not (null x))
       (or (null (car x))
          (AAA (cdr x)))))


2. (defun c1 (x)
(if (consp x)   ; retorna t se o objeto é uma cons
    (c2 (car x) 1 (cdr x))
    x))

(defun c2 (elt n lst)
(if (null lst)
    (list (n-elts elt n))
   (let ((next (car lst)))
   (if (eql next elt)
      (c2 elt (+ n 1) (cdr lst))
      (cons (n-elts elt n)
          (c2 next 1 (cdr lst)))))))

(defun n-elts (elt n)
(if (> n 1)
   (list n elt)
   elt))

3. (defun u1 (lst)
   (if (null lst)
      nil
      (let ((elt (car lst))
           (rest ( u1 (cdr lst))))
         (if (consp elt)
            (append (apply #'lo elt) ; sem # no xlisp
                rest)
            (cons elt rest)))))

(defun lo (n elt)
  (if (zerop n)
      nil
      (cons elt (lo (- n 1) elt))))

4. (defun BBBB (n qua)
     (if (zerop n)
        nil
        (cons qua (BBBB (- n 1) qua))))

Respostas
2. > (c1 '(1 1 1 0 1 0 0 0 0 1))
((3 1) 0 1 (4 0) 1)
Ele comprime listas
```

 ©88-09, Pedro Kantek

```
3.  (u1 '((3 1) 0 1 (4 0) 1))
(1 1 1 0 1 0 0 0 0 1)
Ele descomprime as listas

4. (BBBB 3 '(1 2))
((1 2) (1 2) (1 2))
Repete o segundo operando o número de vezes do primeiro operando
```

7.5.6 Desafios

EXERCÍCIO 18 Escreva uma função que receba duas listas e devolva o átomo que mais aparece nas duas listas
(EX1 '(A B C A B A) '(A B B B C)) é B

EXERCÍCIO 19 Escreva uma função que receba um átomo numérico e uma lista também numérica e devolva qual o elemento da lista que é mais próximo do átomo recebido.
(EX2 '(1 4 10 22 30) '5)) é 4

EXERCÍCIO 20 Escreva uma função que receba um átomo numérico e uma lista também numérica e devolva qual a posição do elemento da lista que é mais próximo do átomo recebido.
(EX3 '(1 4 10 22 30) '5)) é 2

EXERCÍCIO 21 Escreva uma função que receba um átomo e uma lista e devolva outra lista, na qual as ocorrências do átomo foram duplicadas.
(EX4 'A '(A B C D E A F)) é (A A B C D E A A F)

EXERCÍCIO 22 Escreva uma função que receba um átomo e uma lista e devolva outra lista, na qual as ocorrências do átomo foram substituídas por uma lista contendo duas ocorrências do átomo.
(EX5 'A '(A B C D E A F)) é ((A A) B C D E (A A) F)

EXERCÍCIO 23 Escreva uma função que receba uma lista e devolva o último elemento da lista. Você pode usar apenas CAR e CDR.
(EX6 '(A B C D E F)) é F

EXERCÍCIO 24 Escreva uma função que calcule o quadrado do quadrado de um número

EXERCÍCIO 25 Escreva uma função que receba dois átomos númericos e devolva a média geométrica entre eles.

EXERCÍCIO 26 Escreva uma função LISP que receba duas listas e devolva t se a primeira tiver mais elementos do que a segunda e nil senão.

7.6 MYCIN

Aproximadamente 500 regras antecedente-consequente constituem o acervo de conhecimento de MYCIN acerca de aproximadamente 100 causas de infecções bacterianas, como por exemplo As regras tem o formato LISP

```
((and (same cntxt infect primary-bacteremia)
   (membf cntxt site sterilesites)
   (same cntxt portal gi)
   (conclude cntxt ident bacteroides tally 0.7))
```

Tradução:
IF 1. the infection is primary bacteremia, AND
2. the site of culture is one of the steriles sites, AND
3. the suspected portal of entry is grastrointestinal tract,
THEN
There is suggestive evidence (0.7) that the identity is "bacteroides"

7.7 Aplicação prática

Como conclusão do assunto LISP, vamos comentar um pacote completo de programação.

7.7.1 Um tradutor simples

Nesta aplicação, um certo arquivo de texto vai ser lido e caso haja batimento entre as palavras do arquivo e um certo dicionário bilíngüe, as palavras serão substituídas pelas ocorrências do dicionário.

A carga do dicionário Nesta função será criada a variável global denominada `*tab*` que conterá uma lista de listas. Nesta segunda lista haverá 2 átomos: a origem e o destino da tradução.

```
(defun cartab ()
   (setq *tab* (list (list 'i 'eu) (list 'he 'ele) (list 'she 'ela)
  (list 'house 'casa) (list 'tree 'arvore) (list 'beautiful 'bonito)
  (list 'like 'gosto) (list 'ugly 'feio)
  (list 'garbage 'lixo)
  (list 'is 'e) (list 'are 'e) (list 'am 'sou) (list 'the 'o)
  (list 'that 'aquele)))
  ; estabelece na variavel *tab* um dicionario ingles-portugues
  ; para chamar, faça (cartab)
)
```

Leitura dos dados O nome do arquivo será passado à função. Esta o lerá e criará uma lista contendo todas as palavras encontradas.

```
(defun leia (x)
   (setq nomefi (open x))
   (setq texto nil)
   (loop
      (setq armaz (read nomefi () 'eof))
      (setq texto (cons armaz texto))
      (when (eql armaz 'EOF) (return ())))
   (close nomefi)
   (setq texto (cdr texto))
   (reverse texto)
   ; le um arquivo de texto e devole uma lista de palavras
```

 ©88-09, Pedro Kantek

```
   ; para chamar (setq lista (leia "f:arquivo.ent"))
)
```

Busca e troca de uma palavra

```
(defun pesqpal (pa tra)
  (cond
    ((null tra) pa)
    ((eql pa (car (car tra))) (car (cdr (car tra))))
    (t (pesqpal pa (cdr tra))))
    ; pesquisa a palavra no dicionario. Se achar, devolve a tradução. SE
    ; não achar, devolve a propria palavra pesquisada
    ; para chamar (pesqpal 'amigo *tab*)
)
```

A tradução do arquivo todo

```
(defun traduz (li)
   (cond
     ((null li) nil)
     (t (cons (pesqpal (car li) *tab*) (traduz (cdr li)))))
   ; traduz uma lista de palavras. Devolve a lista traduzida
   ; para chamar (traduz '(ouviram do ipiranga as margens))
)
```

Gravação do resultado

```
(defun grava (coisa onde)
   (setq espaco " ")
   (setq nome (open onde :direction :output))
   (loop
     (princ (car coisa) nome)
     (princ  espaco  nome)
     (setq coisa (cdr coisa))
     (when (null coisa) (return ())))
   (close nome)
   ; grava a lista fornecida, mas sem os parenteses,
   ; no local indicado por onde
   ; para chamar (grava (traduz lista) "f:arqs.txt")
)
```

O ciclo completo

```
(defun testa ()
   (grava (traduz (leia "f:arqe.txt")) "f:arqsn.txt")
   ; este e o comando completo que testa tudo o que se fez
)
```

Reversão do dicionário

```
(defun revdic (li)
  (cond
    ((null li) nil)
```

```
    (t (cons (reverse (car li)) (revdic (cdr li)))))
    ; este comando reverte o dicionario (de ing-por para por-ing)
    ; para chamar (setq dicporing (revdic *tab*))
)
```

7.7.2 Um censurador de texto

Aqui, um novo conjunto de funções será criado para processar textos e substituir palavras a censurar (constantes em uma lista) pela palavra ***censurado***.

Cria a lista de palavras censuradas

```
(defun listcens ()
  (setq *cen*
      (list 'curitiba 'parana 'desastre 'morreram 'morreu 'desgosto))
  ; cria na variavel global *cen* uma lista de palavras censuradas
  ; para chamar (listcens)
)
```

Busca a censura a uma palavra

```
  (defun censpal (pa tra)
  (cond
    ((null tra) pa)
    ((eql pa (car tra)) '***censurado***)
    (t (censpal pa (cdr tra))))
    ; pesquisa a palavra "pa" na lista simples "tra".
    ; Se achar, retorna a msg censurado
    ; se não, retorna a propria palavra pesquisada.
    ; para chamar (censpal 'curitiba *cen*)
)
```

Censura o texto todo

```
(defun censura (li)
   (cond
     ((null li) nil)
     (t (cons (censpal (car li) *cen*) (censura (cdr li)))))
     ; censura uma lista (frase) completa
     ; para chamar (censura '(ouviram do ipiranga as margens placidas))
)
```

 ©88-09, Pedro Kantek

Capítulo 8

Sistemas de Produção

8.1 Introdução

Trata-se de um modelo de computação que – ao guardar importante semelhança com a maneira humana de processar dados – tem servido de paradigma em implementações de IA. Um sistema de produção fornece um modelo com controle adequado para o processo de solução de problemas.

Um sistema de produção comporta 3 componentes

1. Um conjunto de regras de produção que representam o conhecimento do sistema
2. Uma memória de trabalho que registra o estado inicial do sistema, do mundo que o envolve e todos os estados intermediários e eventualmente do final
3. Um cliclo de controle do tipo *reconhece - atua*

Regras de produção

Muitas vezes estas regras são chamadas apenas de *produções*. Cada uma das regras é um par de componentes. O primeiro é chamado *condição* e o segundo *ação*. A dupla forma então uma regra cujo formato é *se... então...*. Qualquer semelhança com o onipresente comando *if* de qualquer linguagem de progração NÃO é coincidência.

A condição determina quando esta regra pode se aplicar à solução do problema que se está estudando (ou tentando resolver). A ação, determina um passo na busca dessa solução.

Memória de trabalho

Contém uma descrição do estado atual do mundo. Esta descrição é um padrão que é comparado com as condições da base de regras para selecionais quais são as ações adequadas para alterar este mundo (e resolver o problema). Quando a condição de uma regra coincide com o conteúdo da memória de trabalho, a ação equivalente a esta condição PODE ser aplicada sobre o mundo. As regras de produção são desenhadas de maneira a alterar explicitamente o conteúdo da memória de trabalho.

Ciclo reconhece-atua

É a estrutura de controle de um sistema de produção. A memória de trabalho é inicializada com a descrição inicial do problema. O estado atual do mesmo é mantido através de determinados padrões que sejam significativos para a solução do problema. Estes

padrões são comparados com as condições das regras. As regras para as quais houver "casamento" (entre condição e situação da memória de trabalho) formam um conjunto (chamado de conjunto de conflito). Todas estas ações poderiam ser disparadas, já que todas estão **habilitadas**.

Um subcomponente do sistema de produção, chamado resolutor de conflitos analisa o conjunto de conflito e escolhe uma produção, que é então disparada. A ação é aplicada sobre a memória de trabalho e esta é modificada.

Após isto, o ciclo se repete e assim segue até que nenhuma condição seja satisfeita, quando então o sistema pára.

O resolutor de conflitos deve ter uma estratégia para selecionar a condição elegida. Essa estratégia pode ser simples (escolher a primeira, escolher uma qualquer...) ou pode ser tão complexa quanto se queira, envolvendo heurísticas complexas para a seleção da regra. Este é o modo pelo qual um sistema de produção adiciona heurística a um algoritmo de busca.

Um exemplo que deve ser citado de sistema de produção bastante engenhoso, e que acabou dando origem a um capítulo importante na IA (a computação evolutiva) é o conceito de **sistemas classificadores**.

Propostos por John Holland, eles formam a base para a área de **Vida artificial** ao equiparem os ANIMATS (animat = animal + robot). Trata-se de um sistema complexo de sensores, processadores e atuadores. Os processadores trabalham na modalidade de um sistema de produção e a estratégia de aprendizado e de solução de conflitos é o de um algoritmo genético. As mensagens que trafegam no sistema sofrem os operadores evolutivos: seleção, recombinação e mutação. Mediante o conceito de fitness e do conseqüente algoritmo de distribuição de créditos, o sistema (animat) aprende. Veja mais detalhes sobre este tipo de sistema na aula correspondente de computação evolutiva.

Um modelo puro de SP não possui mecanismo para se recuperar de becos sem saída que surjam na busca, ele simplesmente continua na busca até que mais nenhuma produção esteja habilitada e então pára. Muitas implementações práticas permitem em tais situações a regressão até um estado anterior da memória de trabalho. Este algoritmo que foi aqui descrito é denominado **busca para a frente**.

Seja um exemplo (extraído de [Lug02] pág 177) de um sistema de produção para ordenar uma seqüência alfabética composta pelas letras **a**, **b** e **c**.

Conjunto de regras

1. ba → ab
2. ca → ac
3. cb → bc

A memória de trabalho começa com **cbaca**. Neste exemplo, uma produção é habilitada se a sua condição casar com uma porção da seqüência que se encontra na memória de trabalho. Quando uma regra é disparada, a subseqüência que casou com a condição da regra é substituída pela seqüência que se encontra do lado direito da regra .

Sistemas de produção formam um modelo geral de computação que pode ser programado para fazer tudo o que pode ser feito com um computador. Seu verdadeiro poder, contudo, consiste em ser uma arquitetura para sistemas baseados em conhecimento.

Nº da iteração	memória de trabalho	conjunto conflito	regra disparada
0	cbaca	1,2,3	1
1	cabca	2	2
2	acbca	2,3	2
3	acbac	1,3	1
4	acabc	2	2
5	aacbc	3	3
6	aabcc	0	Parar

A idéia para o projeto baseado em "produção" para a computação tem sua origem nos artigos do lógico Emil Post (1943). A base desta teoria é um conjunto de regras para reescrita de seqüências de caracteres de várias formas similares ao do exemplo acima. Este modelo é equivalente em poder ao de uma máquina de Turing.

Uma aplicação interessante de regras de produção para modelar a cognição humana é encontrada no trabalho de Newell e Simon no que hoje é a Carnegie Melon University nos anos 60 e 70. Os programas que eles desenvolveram incluindo o *General Problem Solver* são em grande parte responsáveis pela importância dos sistemas de produção na IA. Nessas pesquisas as pessoas eram monitoradas em várias atividades de solução de problemas (lógica de predicados, xadrez, etc). O protocolo (padrões de comportamento) dessas pessoas era registrado e decomposto em seus itens elementares. Esses componentes eram os *bits* básicos do conhecimento para resolver problemas e eram dispostos em um grafo (chamado grafo de comportamento do problema). Um sistema de produção era usado para conduzir a busca dentro desse grafo.

Note-se que os pesquisadores usaram um sistema de produção não apenas para conduzir a busca, mas sobretudo como um modelo real do comportamento humano durante a solução de problemas. As produções correspondiam às habilidades das pessoas e estavam registradas na memória de longo prazo dessas mesmas pessoas. Essas produções não são afetadas pela execução do sistema (e o aprendizado ?) e são apenas invocadas.A memória de trabalho corresponde à memória de curto prazo nos seres humanos e descreve o estágio atual da solução de uma ocorrência do problema. O conteúdo da memória de trabalho, em geral, não é preservado após um problema ter sido resolvido. Esta pesquisa é descrita no livro *Human Problem Solving*, de 1972 dos dois autores.

Sistemas especialistas modernos têm perdido um pouco essa aderência original ao comportamento humano, mas o paradigma de sistemas de produção parece adequado para registrar e manter o conhecimento, graças às seguintes características:

- estrutura modular das regras
- separação entre conhecimento e controle
- separação entre memória de trabalho e conhecimento sobre a solução

8.1.1 Tarkin revisitado

Veremos agora o mesmo problema já resolvido com A* sendo tratado através de um sistema de produção Vamos definir as regras:

condição	ação
estado objetivo na memória de trabalho	parar
vazio não está na borda esquerda	mover o vazio para a esquerda
vazio não está na borda superior	mover o vazio para cima
vazio não está na borda direita	mover o vazio para a direita
vazio não está na borda inferior	mover o vazio para baixo

 ©88-09, Pedro Kantek

Regime de controle proposto

- Tentar cada produção na ordem
- Não permitir laços
- parar quando o objetivo for encontrado

Um exemplo, tirado de Nilsson (1980) é mostrado na figura 8.1

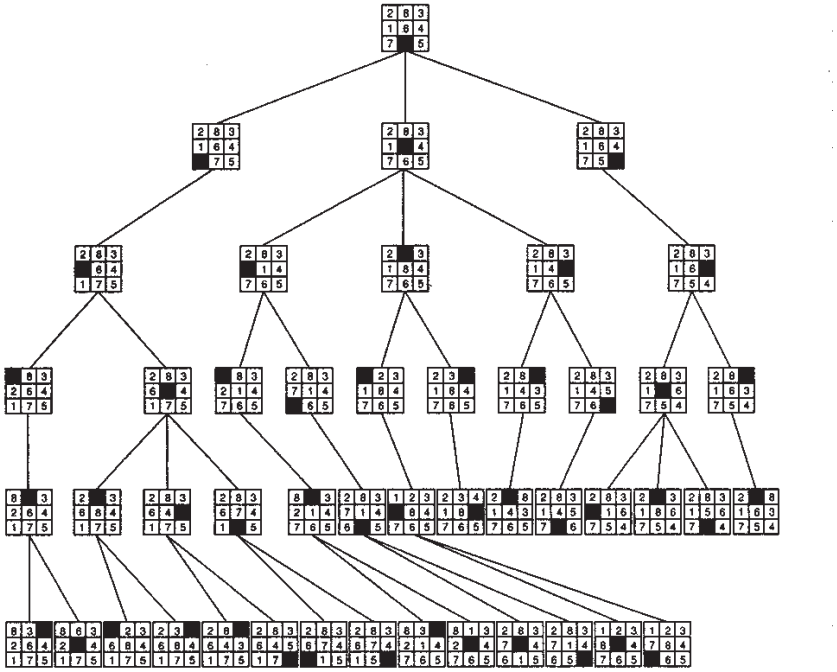


Figura 8.1: Um exemplo tirado de Nilsson 1980

8.2 DENDRAL

Construído em 1969 na Univ. de Stanford, este foi o primeiro representante do novo enfoque. Ele infere a estrutura molecular de uma substância desconhecida a partir de dados obtidos no espectrômetro de massa. O programa recebia a fórmula elementar da molécula (por exemplo $C_6N_{13}NO_2$) e dados do espectrômetro de massa. Por exemplo, um espectro com ponto de máximo em m=15, corresponderia a um fragmento metilo (CH_3). Na primeira versão do programa, dada a entrada ele gerava todas as possibilidades e depois verificava o batimento com a espectrometria obtida. É claro que esta abordagem não funcionava para moléculas grandes (aí está a explosão combinatória, de novo). Para resolver isto, a versão 2 foi alimentada com conhecimento sobre possíveis subgrupos da molécula, como por exemplo o grupo cetona ($C = O$). Eis a seguir a regra que identificava este grupo

se existem picos em x_1 e x_2, tais que **então**
(a) $x_1 + x_2 = M + 28$ (onde M é a massa de toda a molécula)
(b) $x_1 - 28$ é ponto de máximo
(c) $x_2 - 28$ é ponto de máximo

(d) x_1 ou x_2 é ponto de máximo
então existe um grupo cetona
fimse

A importância do DENDRAL é que ele funcionou adequadamente em casos reais e foi o primeiro nisto. Serviu também para mostrar que era possível separar o conhecimento (regras) do raciocínio (motor de inferência), simplificando a tarefa de criar novos sistemas.

8.2.1 Como funciona

Suponhamos que um químico quer saber a natureza química de alguma coisa que acabou de ser criada em um tubo de ensaio. O primeiro passo é determinar a quantidade de átomos de cada elemento na molécula da substância. O resultado é uma fórmula como por exemplo $C_8H_{16}O$. A notação indica que cada molécula tem 8 átomos de carbono, 16 de hidrogênio e um de oxigênio. A próxima etapa é fazer um espectrograma para estudar a forma como os átomos estão dispostos na estrutura da substância. Um espectrograma é produzido como indicado na figura 8.2. A máquina de espectrograma bombardeia uma

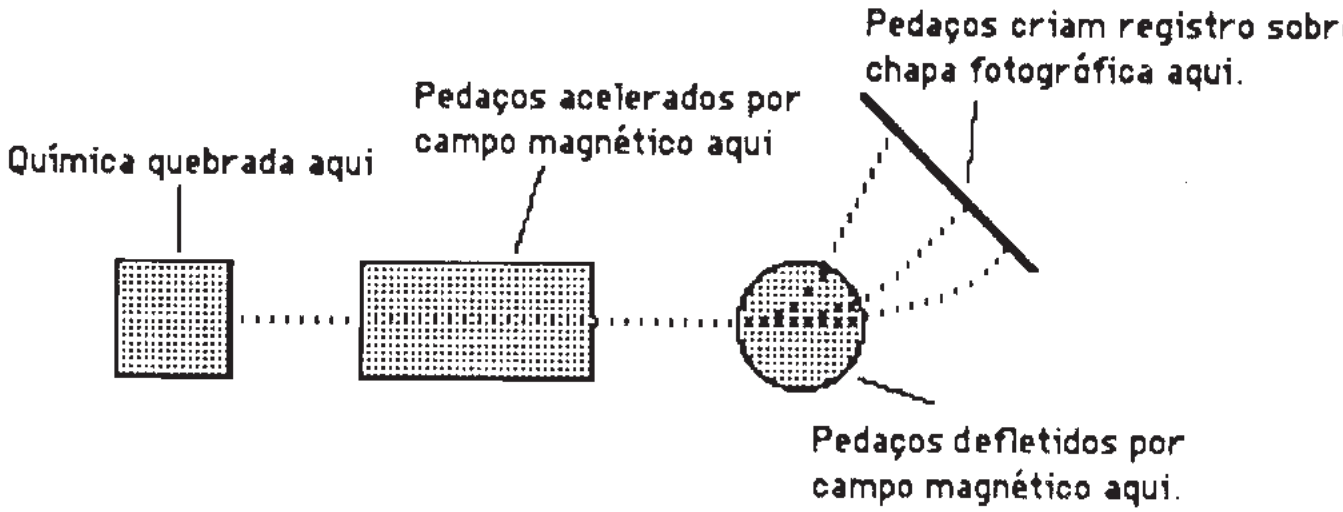


Figura 8.2: Um espectrograma de massa

amostra com elétrons de alta energia que quebram a molécula em pedaços. Cada um dos pedaços tem seu tamanho específico e está carregado eletricamente. Os pedaços são ordenados através de seu deslocamento em um campo magnético que deflete os de carga alta e peso pequeno mais do que os de pequena carga e alto peso. Os pedaços defletidos são recolhidos formando o espectrograma.

A figura 8.3 mostra um caso real, onde o DENDRAL atuando como um químico experiente chegou à formula mostrada

A forma de DENDRAL trabalhar é gerar inúmeras moléculas a partir das regras químicas que o formam e da fórmula básica (atômica) da substância em análise. Para cada molécula, um espectrograma sintético é gerado e comparado com o espectrograma real. As regras de DENDRAL também são usadas para hierarquisar os compostos candidatos.

 ©88-09, Pedro Kantek

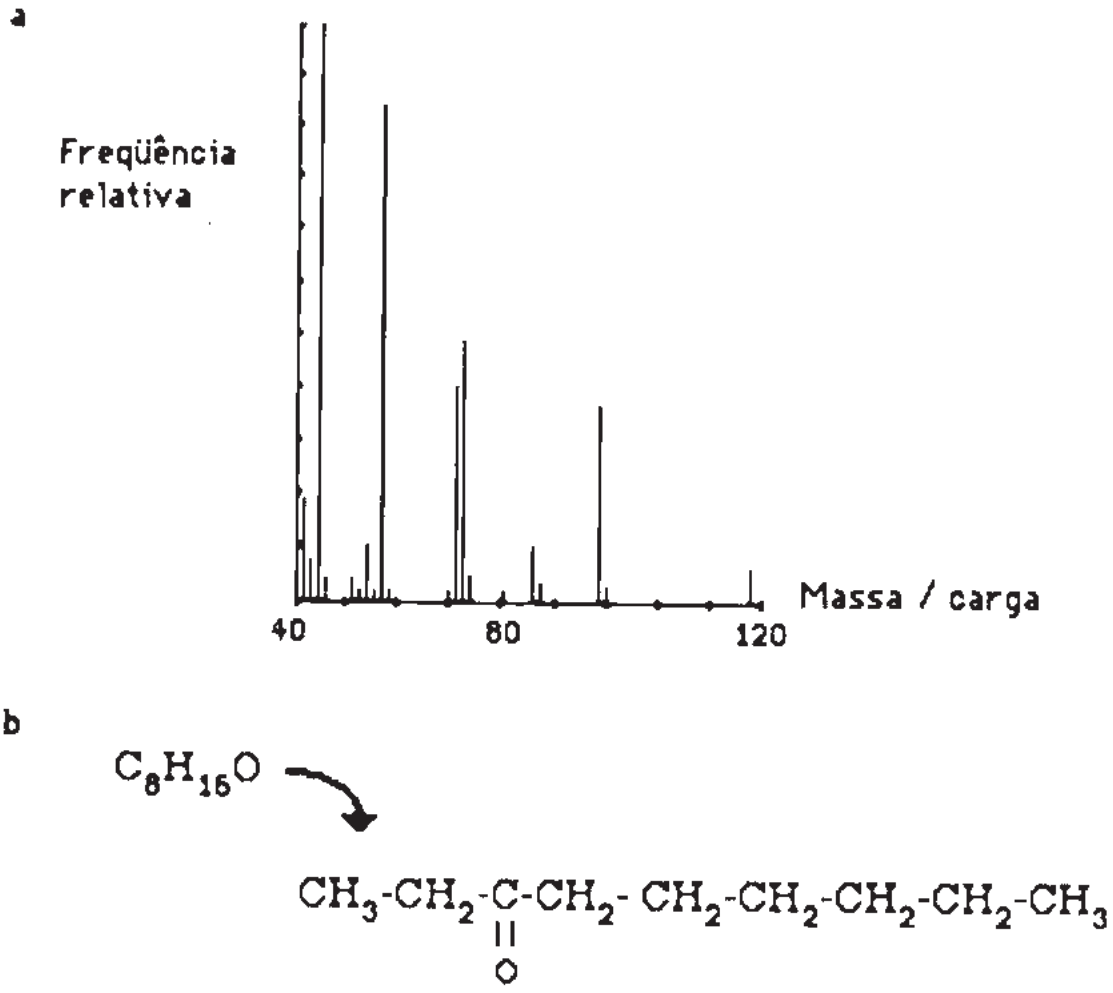


Figura 8.3: DENDRAL trabalhando

8.2.2 MYCIN

Este sistema simula um especialista em infecções sangüíneas. Chegou a ter cerca de 450 regras. Seu desempenho eram bem razoável: empatava com especialistas humanos, mas ganhava sempre de médicos recém graduados. Ele possuia duas diferenças significativas sobre o DENDRAL

- Não contava com um arcabouço teórico (papel feito pela química no DENDRAL). O Mycin se baseava apenas no conhecimento médico.

- Exatamente por isso, ele tratava adequadamente a incerteza.

A inovação apresentada pelo Mycin foi a introdução de coeficientes de certeza, para tratar a falta de conhecimento. Assim, os coeficientes de certeza variam entre -1 (totalmente falso) até 1 (totalmente verdadeiro). Se o coeficiente fica entre -0.2 e 0.2 o fato é desconhecido.

Eis um exemplo de regra dado ao Mycin:
Se o local da infecção é o sangue E
o organismo é gramnegativo E
tem a forma de bastonete E
o paciente pertence a um grupo de risco
então é provável (0.6) que o organismo seja o pseudomonas aeruginosa

Veja um exemplo

Agora é hora de olhar um exemplo de um sistema de análise baseado em regras real.

Geralmente é necessário começar tratamento antibiótico para infecções bacterianas sem saber exatamente que ORGANISMO é o culpado. A razão é que a evidência da CULTURA de laboratório definitiva se acumula demasiado devagar. Para quem está desesperadamente doente, a terapia deve começar imediatamente, não daqui a dois dias. Isso deixa duas opções: o médico prescreve medicamentos de largo espectro para cobrir todas as possibilidades; ou o médico pode prescrever melhor, medicamentos específicos para a doença se ele ou ela, têm o treinamento especial requerido para tirar proveito das evidências disponíveis.

MYCIN se informa a respeito de casos particulares pela solicitação de informações acerca dos sintomas do paciente, condições gerais, histórico, e testes de laboratório fáceis e prontamente obtidos. A cada ponto, a pergunta que MYCIN faz é determinada pela hipótese vigente de MYCIN e pelas respostas a todas as perguntas anteriores. Assim as perguntas começam como se tomadas de uma lista de verificação, mas elas variam à medida que a evidência é construída. Aqui está um extrato de uma sessão exemplo:

```
Qual é o nome do paciente?
   João Silva.
Homem ou mulher?
   Homem.
Idade?
   Ele tem 55.
Você obteve culturas positivas que indicassem o tipo geral?
   Sim.
Que tipo de infecção é?
   Bacteremia primária.
Quando apareceram os primeiros sintomas?
   A 5 de maio.
Chamemos à CULTURA positiva mais recente CULTURA-1. De que local foi
tomada a CULTURA-1?
   Do sangue.
Quando?
   A 9 de maio.
Chamemos ao primeiro ORGANISMO significativo dessa CULTURA ORGANISMO-1.
Conhece a identidade do ORGANISMO-1?
   Não.
O ORGANISMO-1 é um bastonete, um coccus, ou algo mais ?
   Bastonete
Qual é a mancha de Gram do ORGANISMO-1?
   Gram negativo.
Houve culturas positivas de locais que não estão associados com
infecções a respeito das quais você está procurando conselho?
   Não.
```

 ©88-09, Pedro Kantek

```
O ORGANISMO-1 cresceu aerobicamente?
   Sim
O ORGANISMO-1 cresceu anaerobicamente?
   Sim.
Qual é a suspeita de portal de entrada do ORGANISMO-1?
   O trato gastrintestinal.
```

Quando acredita que tem um apanhado razoavelmente bom da situação, MYCIN inicia uma pequena série de perguntas preparatórias para recomendar uma direção de tratamento:

```
Minha recomendação será baseada na opinião de que a identidade
do ORGANISMO-1 pode ser:
1. Pseudomonas-aeruginosa;
2. Klebsiella-pneumoniae;
3. E. Coli;
4. Bacteroides-fragilis;
5. Enterobacter ou
6. Proteus-non-mirabilis.

Em uma escala de 0 a 4, onde o número mais alto indica maior
severidade, como você classificaria o grau de doença
do paciente?
   Grau 3.
O paciente tem alguma reação alérgica clinicamente significativa
a algum agente antimicrobiano?
   Não.
Quanto pesa o paciente?
   70 quilos.

Minha primeira recomendação é a seguinte: a fim de cobrir os itens
1, 2, 3, 5 e 6, dê gentamicina usando uma dose de 119 mg (1,7 mg/kg)
IV, cada 8 h.  Ou IM, por 10 dias. Modifique a dose em caso de problema
renal. E a fim de cobrir o item 4, dê clindamicina usando uma dose de
595 mg (8,5 mg/kg)  q6h IV ou IM, por 14 dias. Se se desenvolverem
diarréia ou outros sintomas gastrintestinais, verifique se é colite
pseudomembranosa.
```

Com esta classe de auxílio, médicos ocupados poderiam considerar mais fatores e fornecer tratamento mais especializado.

Aproximadamente 500 regras antecedente-consequente constituem o acervo de conhecimento de MYCIN acerca de aproximadamente 100 causas de infecções bacterianas, como por exemplo As regras tem o formato LISP

```
((and (same cntxt infect primary-bacteremia)
   (membf cntxt site sterilesites)
   (same cntxt portal gi)
   (conclude cntxt ident bacteroides tally 0.7))
```

Tradução:
IF 1. the infection is primary bacteremia, AND
2. the site of culture is one of the steriles sites, AND
3. the suspected portal of entry is grastrointestinal tract,
THEN
There is suggestive evidence (0.7) that the identity is ”bacteroides”

8.3 Fatores de certeza

Os sistemas baseados em regras usados para identificação, geralmente trabalham em domínios onde as conclusões são raramente indiscutíveis. Assim quem desenvolve sistemas baseados em regras, freqüentemente constrói certa espécie de procedimento de cálculo da certeza por cima do maquinismo básico de antecedentes-conseqüentes. Geralmente, os procedimentos de cálculo da certeza associam um número entre 0 e 1 a cada fato. Este número, chamado de fator de certeza, se destina a refletir quão certo o fato é, 0 indicando um fato definitivamente falso e 1 indicando um fato definitivamente verdadeiro.

Como o cálculo dos fatores de certeza é de importância prática, façamos um desvio para considerar os procedimentos existentes. Compreendamos, en- tretanto, que nenhum desses procedimentos é completamente satisfatório. Vamos ver o que você pode fazer agora e não o que você quer fazer.

Observe que qualquer procedimento para cálculo de fatores de certeza deve incorporar respostas para três questões. Primeira, como se associam as certezas com os antecedentes de uma regra para serem combinados na certeza de entrada final da regra? Segunda, como a própria regra traduz a certeza de entrada na certeza de saída? E terceira, como é determinada a certeza de um fato quando os conseqüentes de várias regras antecedente-conseqüente disputam por ela, requerendo o cálculo de uma certeza multiplamente disputada?

Um procedimento simples, ad hoc, responde as questões desta maneira:

A menor certeza dentre as associadas com cada um dos antecedentes da regra torna-se a certeza de entrada final da regra. (a resistência de uma corrente equivale à resistência de seu elo mais fraco).

Cada descrição de regra inclui como usar um fator de atenuação, o qual, como, os fatores de certeza, varia de 0 a 1. Para calcular o fator de certeza de saída de uma regra, a certeza de entrada é multiplicada pelo fator de atenuação. Exemplo: Se a certeza de entrada é de 0.5 e a regra tem um fator de atenuação de 0.8, a certeza de saída é 0.4.

Havendo várias regras que favorecem algum fato particular, a certeza final atribuída a esse fato é a maior das certezas propostas pelas regras que o favorecem. Assim, a força de um fato é afetada somente pela mais forte das regras que o apóiam. Exemplo: se em uma regra entra uma certeza de 0.9 e também uma certeza de 0.25 a certeza de saída é de 0.9.

Na figura 8.4 mostram-se todos estes conceitos.

Há outros métodos, um deles tirado da estatística, que exige alguma familiaridade com a teoria das probabilidades para ser compreendido.

8.3.1 Outros

O seguinte foi um programa de nome PROSPECTOR escrito com a finalidade de auxiliar na análise de dados geológicos com vistas a mineração. O programa ficou famoso ao recomendar a lavra em um determinado sítio no qual foi descoberta uma jazida de Molibdênio.

Logo no retorno da Apolo 11, trazendo pedras lunares, foi criado o programa LUNAR que respondia perguntas feitas em inglês sobre as pedras. Este programa ficou famoso por ter sido o primeiro PLN a ser usado por outros que não o seu autor.

Como subprodutos destas iniciativas pode-se citar o desenvolvimento da linguagem PROLOG (adequada para a representação e manuseio da lógica de predicados) e o surgimento do formalismo denominado FRAMES por Marvin Misnky.

 ©88-09, Pedro Kantek

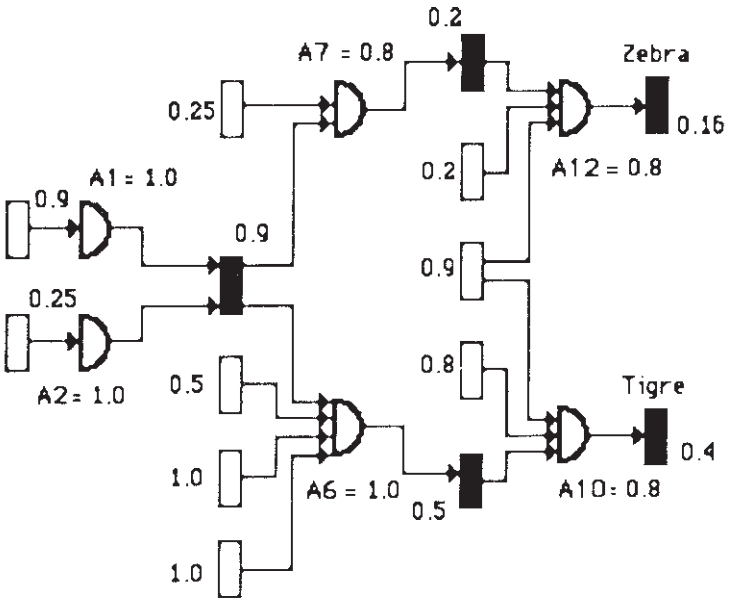


Figura 8.4: certezas de entrada e de saída

Na indústria, em 1982 começou a ser usado o programa R1, construído na Digital. Ele tinha a finalidade de ajudar na configuração de sistemas. A empresa divulgou em 1986 que o uso do programa estava economizando 40 milhões de dólares ao ano.

Em 1981, o governo japonês criou uma iniciativa conjunta (governo, empresas e universidades) à qual foi dado o nome de *Computador de Quinta Geração*. Era um esforço de 20 anos, pretendia-se elevar o paradigma das aplicações de informática embarcando nelas componentes de IA. Lastima-se informar que o esforço não foi bem sucedido, e em poucos anos deixou-se de falar deste assunto.

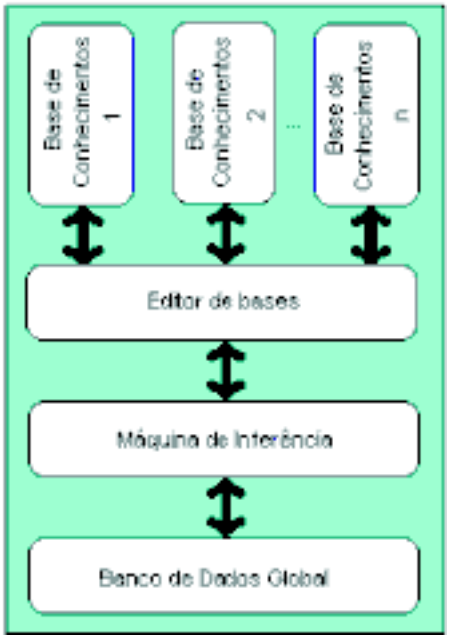
8.4 Um *shell* para sistemas especialistas; ExSinta

Este sistema foi desenvolvido no Laboratório de Inteligência Artificial da Universidade Federal do ceará (www.lia.ufc.br) e é um dos muitos sistemas desta natureza disponíveis na Internet. Contrariamente a diversos sistemas comerciais (que permitem uso limitados de diversos recursos) este aqui está completamente aberto, incluindo-se os seus fontes feitos em Delphi.

Para instalar o sistema, basta baixar o pacote em um diretório qualquer e depois expandir o arquivo comprimido de nome SINTA32.ZIP em outro diretório. Convém expandir também o arquivo MANUAL.ZIP. Além do software, este procedimento também descompacta uma base de teste para detectar doenças em cajuzeiros.

8.4.1 Arquitetura

Os sistemas especialistas gerados usando o EXSINTA seguem a arquitetura



onde

base de conhecimentos representa a informação (fatos e regras) que um especialista utiliza, representada computacionalmente;

editor de bases é o meio pelo qual a shell permite a implementação das bases desejadas;

máquina de inferência é a parte do SE responsável pelas deduções sobre a base de conhecimentos;

banco de dados global são as evidências apontadas pelo usuário do sistema especialista durante uma consulta.

O EXSINTA oferecer o mecanismo de encadeamento para trás *backward chaining*. O encadeamento para trás destaca-se em problemas nos quais há um grande número de conclusões que podem ser atingidas, mas o número de meios pelos quais elas podem ser alcançadas não é grande (um sistema de regras de alto grau de fan out), e em problemas nos quais não se pode reunir um número aceitável de fatos antes de iniciar-se a busca por respostas. O encadeamento para trás também é mais intuitivo para o desenvolvedor, pois é fundamentada na recursão, um meio elegante e racional de programação, para onde a própria Programação em Lógica se direcionou. Em nenhum momento, porém, deixa-se de reconhecer que o encadeamento para frente (foreward chaining) possui vantagens em determinadas ocasiões.

8.4.2 Formato das regras

A regra para o exsinta obedece ao seguinte esquema:

```
SE antecedente ENTÃO <atributo> = valor <grau-de-confiança>
```

O antecedente, por sua vez, obedece ao esquema:

```
<conectivo> <atributo> <operador> <valor>
```

 ©88-09, Pedro Kantek

O conectivo pode ser E, OU ou NÃO. Não há parênteses na regra. Se ambos (E e OU) forem usados na mesma regra, deve-se dizer o EXSINTA qual dos dois é prioritário, e esta informação valerá para todas as regras. Operador pode ser "=" ou "<>" para atributos mono ou multivalorados, e pode ser ">", "<", ">=" ou ">=", além dos dois anteriores para atributos numéricos. Os atributos e valores deverão ter sido criados antes da escrita das regras que os manuseiam. Finalmente, o grau de confiança varia entre 0% e 100%.

Vale lembrar que o EXSINTA implementa o conhecimento monotônico. Isto implica que novas regras nunca tornarão a base constraditória ou inconsistente.

8.4.3 Fatores de confiança

Inicialmente a constação de que seres humanos não costumam trabalhar com intervalos de confiança numéricos. Eles têm dificuldade em se expressar dessa maneira, mas esta é uma boa aproximação para implementar sistemas especialistas em computador. A primeira iniciativa aqui, foi a do MYCIN, que acabou sendo seguida por boa parte das implementações posteriores.

Quando uma premissa é estabelecida por uma confiança c_1 e existe uma regra que usa essa premissa e gera uma conclusão com confiança c_2, então – neste caso – a conclusão será obtida com a confiança $c_1 \times c_2$. Veja-se no exemplo:

Suponha-se a regra

```
REGRA 1
   SE onde estou = centro
   E quanto dinheiro tenho = pouco
   E quer ir = couto pereira
   ENTÃO como vou = a pé CNF 80%
```

Se eu informar, na execução que estou no centro com 70% de confiança, e que tenho pouco dinheiro (100%) e que quero ir ao Couto Pereira (100%), então o sistema manda eu ir a pé com 56% de confiança. Este número é resultado de $0.80 \times 0.70 = 0.56$

Se duas premissas conectadas com "E" tem confianças c_1 e c_2 então a confiança da conjunção será também $c_1 \times c_2$.

Se duas premissas estão conectadas com "OU" e tem confianças c_1 e c_2 então a confiança da disnjunção será $c_1 + c_2 - (c_1 \times c_2)$.

Finalmente, se uma mesma variável recebe confianças diferentes em momentos também diferentes, o cálculo final da confiança usa a mesma regra da disjunção acima vista.

Por exemplo: Suponha que no início do processamento se estabelece que estou no centro com 90% de confiança. Mais adiante, uma regra chega a conclusão de que estou no centro com 50%. A partir deste ponto a confiança de que onde estou = centro é de $0.9 + 0.5 - (0.9 \times 0.5) = 1.4 - 0.45 = 0.95$.

Deve-se lembrar que o sistema considera uma igualdade verdadeira quando a confiança é maior ou igual a 50% (se bem que este valor pode ser globalmente alterado).

O EXSINTA considera todas as variáveis podendo ter o valor DESCONHECIDO.

8.4.4 Implementação

As bases geradas são salvas como arquivos com tipo = BCM.

Para implementar um SE, deve-se:

1. Definir as variáveis, dizendo para cada uma, seu nome, seu tipo (monovalorada, multivalorada ou numérica) e seus possíveis valores. Variáveis numéricas não têm valores pré-definidos, mas precisam ter intervalos de existência. (a;b = entre a e

b, ;b = maior ou igual a b, a; menor ou igual a a). Se uma variável não tem valor, EXSINTA assume que ela é lógica (sim/não).

Apagar uma variável exclui as regras que a manuseiam.

Para proceder a esta etapa, abra uma base (existente ou nova), e clique no botão variável. Clique em adicionar variável e depois em adicionar valores. Neste último caso, uma variável deve estar selecionada.

2. Definir objetivos. Aqui se declara o que o sistema especialista deve fazer (ou melhor dizendo, para quais variáveis (originalmente desconhecidas) o SE deve encontrar valores. Para realizar esta etapa deve-se pressionar o botão OBJETIVOS. Em seguida o sistema apresentará duas listas: a de variáveis e a de objetivos, esta inicialmente vazia. As duas listas podem intercambiar variáveis.

   Note-se que se o sistema for rodado sem nenhum objetivo ele não vai fazer nada.

3. Criar regras. Clicar em NOVA REGRA e preencher as informações pedidas. É conveniente dar nomes às regras, para possibilitar referências (humanas) posteriores. Lembrar que um SE de verdade chega facilmente às centenas de regras. Pode-se usar regras anteriormente definidas como modelos, isto acelera o trabalho.

   Finalmente, deve-se notar que a ordem das regras é importante para o EXSINTA. A ordem pode ser modificada arrastando as regras na lista original.

4. Rodar o SE. Finalmente, pode-se rodar o sistema, apertando-se a tecla de RUN na barra de ícones. Note-se que a cada consulta realizada, o sistema – além de responder – ainda mostra o histórico da consulta, os valores das variáveis envolvidas e as regras originais, bastando selecionar a aba correspondente na saída.

8.5 Implementação de um SBC

Um agente baseado em conhecimento tem como componente fundamental a *base de conhecimento*. Informalmente, a BC é um conjunto de representações sobre certa parte do mundo. Cada componente da BC é chamada *oração* e estas estão representadas em uma linguagem para representação do conhecimento.

A BC precisa ser populada com orações. A operação de incluir uma nova oração à BC é chamada genericamente de *informar*, e o processo de interrogar a BC a respeito de alguma coisa é chamado gerericamente de *perguntar*. A ligação entre ambos os processos está em que ao perguntar alguma coisa à base, a resposta obrigatoriamente terá que estar de acordo com os fatos que foram anteriormente informados a ela.

A obtenção de respostas às perguntas efetuadas também é responsabilidade do segundo componente de agentes baseados em conhecimento que é o *mecanismo de inferência*.

O algoritmo em alto nível de um agente pode ser

Entrada: percepção
Saída: ação

```
1: função AGENTE-BASEADO-CONHECIMENTO
2: BC: base de conhecimentos
3: t: global, relógio, setado originalmente em 0
4: INFORMAR (percepção, t)
5: ação ← PERGUNTAR (BC,t)
6: INFORMAR (ação,t)
7: t++
```

 ©88-09, Pedro Kantek

Os detalhes da linguagem de representação do conhecimento estão ocultos no interior das funções INFORMAR e PERGUNTAR. Entretanto, pode-se pensar nesta linguagem em pelo menos 3 níveis:

Epistemológico É o mais abstrato dos 3. Descreve os fatos que o agente conhece em linguagem coloquial.

Lógico Este nível descreve as orações que codificaram o conhecimento acima em linguagem formal

Implementação Este nível opera sobre a arquitetura do agente. Aqui se encontram as representações físicas das orações correspondentes ao nível lógico. Depende da arquitetura da aplicação e tem alguma importância para a eficiência do agente

Embora não obrigatório, é adequado que o agente esteja preparado para ser informado sobre todo o conteudo da BC. Em outras palavras, a BC deveria começar zerada. Isto facilita a armazenagem da BC e é conhecido como *enfoque declarativo* de construção de um sistema, sendo altamente desejável. Note-se que este enfoque não impede que o agente possua mecanismos de aprendizagem que lhe possibilitem ir além dos fatos informados a ele.

8.5.1 Representação, Raciocínio e Lógica

Uma linguagem para representação do conhecimento possue os seguintes aspectos:

síntaxe Determina as configurações possíveis na construção de orações. Embora principie sobre uma folha de papel em branco, é conveniente pensar na síntaxe como padrões de memória do computador

semântica Correlaciona os fatos do mundo às orações construídas. Sem a semântica as orações seriam apenas padrões na memória ou símbolos no papel sem nenhum significado. Com a semântica, pode-se afirmar que o agente sabe algo sobre o mundo.

Se a semântica e a síntaxe estão definidas de maneira precisa, pode-se afirmar ser esta linguagem uma *lógica*. A inferência serve para criar novas orações. Espera-se que tais orações representem igualmente feitos no mundo e esta é toda a dificuldade de criar agentes racionais.

As novas orações geradas pela inferência devem ser verdadeiras, supondo que as orações prévias (que "entraram"na inferência) também o sejam. A esta relação entre as orações denomina-se *implicação*.

A inferência permite obter duas coisas: Na primeira, pode-se, a partir da BC, através da inferência, gerar novas orações. Na segunda, dada uma BC e uma oração, pode-se perguntar se esta oração é conseqüência da BC ou não. Denomina-se *demonstração* ao processo completo da inferência que permite gerar novas orações.

Suponha que o conjunto de todas as conseqüências da BC é um depósito de palha e uma dada oração α é uma agulha. A implicação é a afirmação de que a agulha está dentro do depósito. A demonstração seria encontrar a agulha.

Um procedimento de inferências é *completo* se é capaz de encontrar a demonstração de toda oração implicada. Note-se que em muitas BC o palheiro é infinito, pelo que a característica da completeza é especialmente relevante.

A *teoria da demonstração* é a especificação de passos de raciocínio (ou de inferência) confiáveis para obter novas orações que representem feitos derivados dos feitos representados na BC (e por extensão do mundo, é certo).

Veja-se no seguinte exemplo, extraído da matemática, a síntaxe, a semântica e a teoria da demonstração. Seja a oração $E = mc^2$. Na síntaxe da matemática, entende-se que o símbolo = relaciona duas expressões formadas por constantes, variáveis e operações. Na semântica, afirma-se que as duas expressões valem a mesma coisa. Com base na teoria da demonstração, afirma-se poder concatenar (multiplicando) uma nova constante. Assim, uma nova oração pode ser $ET = mc^2T$

Linguagem *expressiva* é aquela na qual não faltam ferramentas para representar o conhecimento completo. Por exemplo, a linguagem matemática não é expressiva para representar o fato de que está chovendo. Mesmo as linguagens naturais evoluiram no sentido da comunicação e não necessariamente no da representação do conhecimento. Há ambigüidades. Por exemplo pequenos cachorros e gatos. Gatos são pequenos também ou apenas os cachorros.

Características esperadas de uma linguagem de representação do conhecimento

expressiva Deve poder armazenar o conhecimento completo

concisa Ser sucinta, sem desperdícios

eficiente Permitir gerar inferências a partir das orações já armazenadas

composta O significado de uma oração depende do significado de suas partes

Existem infinitas linguagens que podem ser criadas atendendo a estes requisitos. A *lógica de primeira ordem* é a mais usada e a mais estudada para representar conhecimento em IA.

Raciocínio e inferência são sinônimos e eles descrevem o processo pelo qual se obtém conclusões. O raciocínio confiável recebe o nome de *inferência lógica* ou *dedução*. Este processo serve para estabelecer uma relação de implicação entre duas orações.

 ©88-09, Pedro Kantek

Capítulo 9

Programação Lógica

> Programadores de linguagens convencionais (inclusive funcionais) quando introduzidos em programação em lógica sofrem da síndrome da abstinência em relação a comandos de atribuição e outras características orientadas à máquina.. C. J. Hogger, Introduction to logic programming, 1984.

9.1 Histórico

- Precursores: Newell, Shaw e Simon, com sua Logic Theory Machine, que buscava a prova automática de teoremas, em 1956.
- Robinson, 65, no artigo A machine oriented logic based on the resolutrion principle. Propunha o uso da fórmula clausal, do uso de resolução e principalmente do algoritmo de unificação.
- Green em 1969, escreveu o Question Answer System, que respondia perguntas sobre um dado domínio. Na mesma época, Winograd escreveu o software Planner.
- Todos estes tinham problemas de eficiência, por causa, entre outras coisas, da explosão combinatória.
- Em 1970, em Edinbourg, Kowalski bolou o seguinte esquema:
  - Limitou a expressividade às cláusulas de Horn
  - Criou uma estratégia de resolução, não completa, mas muito eficiente
- Em 72, em Edinbourg e Marselha, surgiu a linguagem PROLOG.
- Em 77, Waren escreve o primeiro compilador de PROLOG. Começa o sucesso de PROLOG na Europa. Nos EUA, sempre deu-se maior atenção ao LISP.
- Foi a base para o computador de 5^a geração do Japão.

9.1.1 Cláusulas de Horn

uma cláusula é de Horn, quando tem apenas 1 cláusula no lado direito. Passando a cláusula para a fórma canônica, verifica-se que a cláusula de Horn é um monte de cláusulas ligadas por OU e apenas 1 NÃO negada. A cláusula de Horn intuitivamente corresponde a definir algo.

Exemplo: se (tem garras) E (é preto) E (é carinhoso) E (é charmoso) $\Rightarrow$ (é o binho).

¬ (tem garras E é preto E é carinhoso E é charmoso) OU (é o binho)
¬tem garras OU ¬é preto OU ¬é carinhoso OU ¬é charmoso OU é binho

Para definir o binho como sendo um conjunto de Ous, não poderia usar uma única cláusula de Horn, precisaria usar mais de uma. Daí em cada uma a regra acima se verifica.

9.1.2 Conceitos básicos

O Prolog é um sistema lógico formado por um domínio (conjunto de indivíduos), predicados (relações entre os indivíduos do domínio) e conectivos (E, OU, implica, não e equivale).

Fórmula

Um predicado é uma fórmula. Se F1 e F2 são fórmulas, F1 conectivo F2 é uma fórmula.

Variáveis

símbolos que podem ser qualquer indivíduo. Em prolog, os indivíduos do domínio se escrevem em minúsculo e as variáveis em maiúsculo + underscore. A variável "_"é a chamada variável anônima.

Cláusula

Representação canônica (cânon = regra) de uma expressão lógica. Mediante um "cozimento"(a la Jacques Facon), das fórmulas, elas sempre podem ser trazidas a esta representação, que a menos de sinônimos é única.

É uma cláusula que:

1. Só tem os conectivos $\wedge$ (e), $\vee$ (ou) e $\neg$ (não)
2. Os OU estão dentro dos parênteses e os E estão fora

As fórmulas usadas para chegar à forma canônica são:

Forma original	Forma canônica
$F_1 \iff F_2$	$F_1 \Rightarrow F_2 \wedge F_2 \Rightarrow F_1$
$F_1 \Rightarrow F_2$	$\neg F_1 \vee F_2$
$\neg(F_1 \wedge F_2)$	$\neg F_1 \vee \neg F_2$
$\neg(F_1 \vee F_2)$	$\neg F_1 \wedge \neg F_2$
$F_1 \vee (F_2 \wedge F_1)$	$(F_1 \vee F_2) \wedge (F_1 \vee F_2)$
$\neg\neg F_1$	F_1

Exemplo

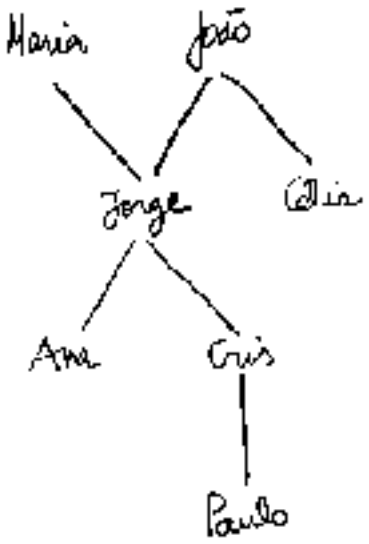
Seja a seguinte arvore genealógica

Podemos escrever:

```
parent(maria,jorge):-true.
parent(joao,jorge):-true.
parent(jorge,ana):-true.
...
parent(cris,paulo):-true.
```

Daí: podemos perguntar:

```
? parent(joao,jorge)
YES
? parent(celia,cris)
NO
```




```
? parent(cris,carlos)
NO
? parent(X,jorge)
X=maria
X=joao
no (significando acabou)
? parent(X,Y)
X=maria
Y=jorge
X=joao
Y=jorge
...
? parent(X,Y),parent(Y,paulo)  [ler esta vírgula como $\lor$]
X=jorge
Y=cris
```

Relações podem ser definidas pela declaração das n-tuplas que satisfazem a relação. Não deixa de ser uma regra degenerada. Quando escrevemos jovem(marcia):-true. estamos dizendo na verdade que marcia é jovem se ... true. Logo é sempre verdade.

As perguntas, que o prolog chama tecnicamente de goals, podem ser um ou vários separados por vírgulas que fazem o papel de E.

Mais alguns exemplos de regras

```
female(maria):-true.
male(joao):-true.
...
offspring(Y,X):-parent(X,Y).
```

Para este tipo de regra (na verdade para qualquer regra) define-se o que vem antes do :- como cabeça (head) da regra, (e que se for de Horn, será uma só), e o que vem depois do :- como corpo (body) da regra. A regra acima deve ser lida como Y é offspring de X se X é parent de Y. Logicamente falando, o certo seria: Se X é parent de Y então Y é offspring de X. Daí que

```
offspring(celia,joao) é true.
```

 ©88-09, Pedro Kantek

Regras com corpo composto Por exemplo

```
mother(X,Y):-parent(X,Y),female(X)
grandparent(X,Z):-parent(X,Y),parent(Y,Z)
sister(X,Y):-parent(Z,X),parent(Z,Y),female(X)
```

Esta última regra não está completa e correta, pois sister(ana,cris) é true, mas sister(ana,ana) também é. Ou seja, teríamos que colocar a informação de que X tem que ser diferente de Y.

Regras recursivas

Aqui está o grande poder de fogo do PROLOG. Vejamos um exemplo

```
predecessor(X,Z):-parent(X,Z)
predecessor(X,Z):-parent(X,Y),predecessor(Y,Z)
```

Note que se define o predicado predecessor em função de predecessor.

9.1.3 Introdução

> Este texto é uma adaptação do texto *Elements of Logic Programming*, de MOSSES, Peter, disponível em http://wiki.daimi.au.dk/dSprogSem-03.

A programação lógica é um dos paradigmas usuais de programação (os outros: imperativa, orientada a objetos e funcional). Ela compartilha com o paradigma funcional a pouca importância dada ao "como", centrando-se no "o que". Na programação funcional, as frases da linguagem são funções com seus domínios e contra-domínios. Aqui as frases são as fórmulas atômicas envolvendo aplicações de predicados e implicações entre tais fórmulas atômicas. Isto dá à programação lógica seu sabor único.

A principal linguagem aqui é o PROLOG. Ele é largamente usado em conexão com bancos de dados, para construir sistemas baseados em conhecimento e engenhos de inteligência artificial. Existem diversas implementações de PROLOG, consulte o anexo 1 para obter algumas delas.

A programação lógica pode ser tão complexa quanto se queira. Nesta aula apenas alguns aspectos básicos serão vistos. O objetivo é entender os conceitos fundamentais da programação lógica e em seguida conseguir escrever programas simples em PROLOG. A sessão 2 trabalha com aplicações envolvendo dados não estruturados. A 3 mostra como a unificação e resolução são usadas para executar programas lógicos e o efeito operacional do "cut". A sessão 4 introduz dados estruturados focando-se em listas. Finalmente, a sessão 5 mostra como gramáticas lógicas podem ser usadas para gerar *parsers* em PROLOG.

9.1.4 Programas

Fatos individuais consistem de fórmulas atômicas que não usam variáveis. O que segue é um programa PROLOG consistindo de uma lista de fórmulas simples descrevendo fatos sobre a existência de arestas dirigidas entre os nodos `a,...,e` de um grafo:

```
aresta(a,b).
aresta(a,e).
aresta(b,d).
aresta(b,c).
aresta(c,a).
```

```
aresta(e,b).
```

Por exemplo, a fórmula atômica (ou átomo, por brevidade) `aresta(a,b)` expressa o fato de que existe uma aresta que vai do nodo `a` para o nodo `b`. A regra simples `aresta(a,b).` assegura tal fato.

Identificadores (como `aresta`) são chamados predicados e eles se aplicam a argumentos de maneira muito similar a como funções se aplicam a argumentos em outras linguagens. A aplicação de um predicado a um argumento em particular expressa que o valor desse argumento está relacionado ao predicado. Essa aplicação não retornará nenhum resultado.

Em PROLOG os predicados são usados sem declaração prévia. Eles são *untyped*. O número de argumentos de um predicado é denominado a sua aridade. No exemplo a aridade de `aresta` é 2. O mesmo identificador pode ser usado para predicados diferentes, desde que não tenham a mesma aridade. O PROLOG distingue-os anexando ao nome do predicado a sua aridade, ou no exemplo `aresta/2`.

Os identificadores tais como `a,...,e` que ocorrem como argumentos de predicados são constantes e tratadas como valores distintos. Duas constantes só são iguais se tiverem o mesmo nome. Note-se que as constantes são simplesmente usadas sem qualquer declaração prévia, ao contrário da maioria das outras linguagens de programação.

Em geral, implementações do PROLOG são incapazes de detectar nomes mal escritos (mal formados) de constantes ou predicados, nem erros de aridade na aplicação de predicados, razão pelo qual cuidado redobrado deve ser dispendido ao escrever programas.

Nomes de predicados sempre começam com letra minúscula, assim como as constantes. Dígitos, tralhas e sublinhados podem ser usados nos nomes. Regras sempre terminam por um ponto (e não por " ; " que desempenha outro papel em PROLOG).

Um nome em PROLOG é uma variável se ele começa com letra MAIÚSCULA. Variáveis também são não-tipadas e podem assumir o valor de qualquer constante usada no programa. Mais tarde veremos outros tipos de variáveis como listas.

Considere a adição no programa anterior da regra, envolvendo a variável `S`.

```
aresta(S,S).
```

A regra acima assegura que para cada nodo no grafo, existe uma aresta que conecta esse nodo a ele mesmo. Isto é um fato universal. A variável `S` pode assumir qualquer um dos valores das constantes `a,...,e`. Se mais tarde introduzirmos outra constante `f` para extender o programa com uma regra envolvendo `f` como seu argumento, `S` assumirá o valor `f` também: a ordem das regras não afeta os valores das variáveis.

As variáveis usadas em uma regra são locais a essa regra, assim não faz diferença se a mesma variável é usada em outras regras ou não. Variáveis diferentes na mesma regra podem (em contraste com constantes) ter o mesmo valor. Por exemplo `aresta(S,T)` garante que para cada nodo há uma aresta para outro nodo (que pode inclusive ser o mesmo).

Um programa PROLOG consiste em uma lista de regras. O programa é executado quando se chama o sistema PROLOG (tarefa essa que varia muito em função do prolog que se usa) e depois carregando o arquivo que contém o programa. Suponhamos que as nossas regras sobre as artestas estejam em um arquivo de nome `grafo.pl`. Então, ele pode ser carregado escrevendo

```
?- [grafo].
```

Onde a constante "?-" é o prompt ("pronto") do sistema.

Depois que o programa foi carregado, o usuário pode especificar *query*s (conhecidas como *queries* ou também como objetivos ou *goals*). Por exemplo, a seguinte *query* se existe uma aresta conectando `a` a `c`:

```
?- aresta(a,c).
```

A resposta a esta *query* é simplesmente `No`, já que este fato não está assegurado

 ©88-09, Pedro Kantek

ao (pelo) programa nem é uma conseqüência dos fatos assegurados. Tivéssemos feito a *query* `aresta(c,a)` e a resposta seria `Yes`. Note que neste exemplo tão simples as respostas são imediatamente óbvias.

Depois disto, o PROLOG está pronto para uma nova *query*. Tentemos uma envolvendo a variável `X` (não faz a menor diferença se esta variável foi ou não foi usada no programa).

```
?- aresta(b,X).
```

Poder-se-ia imaginar que a resposta deveria ser um simples `Yes` ou um `No`, mas o PROLOG faz mais coisas. Ele responde para quais valores de `X` a *query* é respondida satisfatoriamente, neste caso, `X=d`.

A seguir, o PROLOG precisa ser informado como queremos que ele continue a resposta. Teclando um " ; " (ponto e vírgula) estamos dizendo a ele que queremos que ele continue a responder. Fazendo isto, ele responde com `X=c` que vem a ser a segunda resposta (de aresta que sai de `b`). Teclando mais um " ; " ele responde `No` significando que não há nenhum vértice a mais.

Teclando a qualquer momento um *newline* ou *enter* o mesmo aborta a *query* atual e se prepara para responder a uma nova *query*. teclando um *enter* o PROLOG sempre gera um `Yes` como resposta.

Diversos átomos podem ser combinados para gerar uma *query* única. Esta *query* é satisfeita por aqueles valores da variável para os quais todos os átomos resultam verdades. Por exemplo, a seguinte *query* é satisfeita apenas para `X=b`

```
?- aresta(a,X), aresta(X,c).
```

A essência da programação lógica é a possibilidade de derivar fatos implícitos. Aqueles que são conseqüências lógicas dos fatos listados explicitamente no programa. Além da substituição de variáveis e da conjunção, a principal maneira pela qual conseqüências podem ser achadas é de acordo com as regras condicionais:

> Quando todas as condições de uma regra são verdadeiras (para uma dada substituição de variáveis) então a conclusão dessa regra também é verdadeira (para a mesma substituição)

Por exemplo, vamos adicionar a seguinte regra condicional ao nosso programa de caminhos em grafos

```
caminho2(S,T) :- aresta(S,X), aresta(X,T).
```

Isto pode ser lido como: Para todos os valores de `X, S,` e `T` o fato `caminho2` é verdadeiro se ambos os predicados `aresta(S,X)` e `aresta(X,T)` são verdadeiros.

Por exemplo, `caminho2(a,c)` é verdadeiro, já que ambos `aresta(a,b)` e `aresta(b,c)` o são. Mas, `caminho2(c,a)` não é verdadeiro, já que o único nodo para o qual `aresta(c,X)` é verdadeiro é `a` e `aresta(a,a)` não é verdadeiro.

Note que `X` não aparece nas conclusões da regra, ele é tratado apenas nas condições. Para todos os valores de `S` e `T`a conclusão `caminho2(S,T)` é verdadeira se existe um valor para `X` que satisfaça ambas as condições.

Assim como listamos fatos na forma de regras simples, podemos ter mais de uma regra condicional para a mesma conclusão.

```
conectado(S,T) :- aresta(S,T).
conectado(S,T) :- aresta(T,S).
```

Tomadas juntas as duas regras acima devem ser lidas como `S` e `T` estão conectados se existe `aresta(S,T)` **ou** se existe a aresta `aresta(T,S)`.

Tal disjunção de condições para a mesma conclusão pode ser expressa usando um " ; " (ponto e vírgula) ao invés de uma vírgula. Fica

```
conectado(S,T) :- aresta(S,T) ; aresta(T,S).
```

O átomo `caminho2(S,T)` assegura que existe um caminho de comprimento 2 entre os vértices `S` e `T`. Como poderíamos especificar o átomo `caminho(S,T)` que fosse verdadeiro sempre que existisse um caminho, de qualquer comprimento inteiro e positivo de `S` a `T` ? Claramente seria muito tedioso especificar os átomos `caminho3(S,T)`, `caminho4(S,T)`, ...

Pensando indutivamente, existe um caminho de `S` a `T` se

- existe uma aresta entre `S` e `T`, ou
- existe uma aresta de `S` até um nodo intermediário `X` e deste há um caminho até `S`.

Estas duas coisas podem ser especificadas pelas seguintes regras

```
conectado(S,T) :- aresta(S,T).
conectado(S,T) :- aresta(S,X), conectado(X,T).
```

A primeira regra acima é conhecida como o caso básico e a segunda é chamada a regra recursiva, já que o predicado usado na conclusão ocorre também nas condições da regra. O PROLOG avalia esta condição usando a estratégia em profundidade. Quando o grafo em estudo é acíclico, o PROLOG sempre vai encontrar o caminho se este existir. Entretanto, se ele for cíclico, poderemos ter um loop infinito, já que ele vai iniciar novas recursões a partir do nodo originalmente inicial. Uma técnica para evitar este tipo de loop é guardar uma lista da seqüência de argumentos que já foram pesquisados, como ver-se-á no cap. 4.

EXERCÍCIO 27 Proceda às seguintes operações:

1. Instale o software SWI-PROLOG na sua máquina. Ele pode ser buscado em http://www.swi-prolog.org.
2. Copie as regras `aresta`, `caminho2` e `caminho` para um arquivo chamado teste.pl
3. Carregue o SWI-PROLOG e depois carregue o arquivo das regras escrevendo `[teste].`
4. verifique se a *query* atômica `caminho(c,d)` é verdadeira
5. investigue quais pares de nodos substituem `X` e `Y` como resposta à *query* `caminho(X,Y)`. Existe algum par que não é listado e que tem um caminho através do grafo ? Se existir, dê um exemplo.

EXERCÍCIO 28 Proceda às seguintes operações:

1. Especifique um grafo dirigido em PROLOG tal que para os nodos particulares `f` e `g` a *query* atômica `caminho(f,g)` não exista. Dica: 3 nodos e 3 arestas são suficientes.
2. Especifique as *query*s atômicas `guitracer` e `trace` e siga o progresso da pesquisa elaborada acima. Forneça uma explicação da causa da não-terminação baseada em suas observações.
3. Explique porque a não-determinação não aperece quando o grafo é acíclico.

EXERCÍCIO 29 Considere as regras PROLOG para `aresta` e `caminho`, mas com as regras deste último, invertidas, de tal forma que a regra recursiva seja vista antes da regra do caso básico. Forneça exemplos de *query*s envolvendo `caminho` de maneira que:

1. Haja sucesso
2. Haja falha
3. O programa entre em loop

 ©88-09, Pedro Kantek

9.1.5 Cálculos

Vamos considerar novamente, com algum cuidado como os cálculos são feitos em PROLOG. Para simplificar a explicação vamos olhar para regras bem simples: as que tem a lista de condições vazia. O átomo na conclusão é chamado cabeça da regra.

Considere a seguinte *query* `?- caminho(a,X).` É tentador imaginar que o PROLOG tenta satisfazer a *query* fazendo sistematicamente todas as possíveis substituições de `X`. Entretanto, muitas vezes este conjunto de valores pode ser infinito e neste caso, quando a query não é verdadeira, o processo poderia entrar em loop e a resposta `No` nunca surgiria.

Ao contrário, PROLOG tenta satisfazer `caminho(a,X)` sem nenhum substituição a priori de algum valor para X. E, a substituição de X é determinada apenas em conexão com a satisfação da query. A variável que não tem valor é chamada *unbound* ou não-instanciada.

A questão chave aqui é o processo de **unificação**.

Dois átomos são unificáveis quando existe uma (possivelmente vazia) substituição para suas variáveis de modo que os átomos se tornem idênticos. A substituição é chamada unificador para os átomos. O processo de achar a substituição que faça os dois átomos idênticos é chamada unificação. A substituição mínima que torne dois átomos idênticos é chamado o unificador mais geral para eles.

Claramente quando dois átomos são unificáveis eles devem ter o mesmo predicado e o mesmo número de argumentos. Quando uma constante ocorre em um argumento no átomo, o correspondente argumento no outro átomo deve ser a mesma constante ou uma variável que possa ser substituída pela constante.

Para ilustrar considere os dois átomos `caminho2(a,X)` e `caminho2(S,T)`. Qualquer unificador tem que substituir `a` por `S`, mas para fazer o segundo argumento idêntico, ambos os agrumentos `X` e `T` teriam que ser trocados por constantes ou variáveis. Podemos evitar isto fazendo simplesmente a substituição de uma variável pela outra, a saber `T` por `X`. De fato, a substituição de `a` por `S` e de `T` por `X` é o unificador mais geral para estes dois átomos.

EXERCÍCIO 30 Indique quais dos seguintes pares são unificáveis, dando os unificadores quando possível

1. `caminho(A,b)` e `caminho(c,X)`
2. `caminho(A,b)` e `caminho(b,c)`
3. `caminho(X,Y)` e `caminho2(a,b)`

A próxima etapa no trabalho do PROLOG é a **resolução**. Ela troca uma *query* atômica que pode ser unificada com a cabeça de uma regra pelas condições da tal regra.

Lembremos que uma *query* pode ser uma conjunção de átomos. A etapa da resolução é possível quando um átomo de uma *query* em particular é identico ao átomo da cabeça de alguma regra. A *query* atômica é então trocada pelas condições dessa regra. Claramente se estas condições podem ser satisfeitas, então a cabeça da regra também pode. Assim a resolução preserva a satisfatibilidade.

Já que regras que contenham variáveis podem ser usadas para todas as possíveis substituições de valores para essas variáveis pode-se permitir a etapa da resolução mesmo quando uma query atômica é idêntica à cabeça de uma regra após alguma substituição (que obviamente também é aplicada às condições da regra).

Similarmente, pode-se aplicar a substituição à query atômica em si mesma – fazendo com que a mesma substituição seja aplicada a todos os outros átomos na query. Se o

resultado após a substituição pode ser satisfeito como também a query original, de novo, isto preserva a satisfatibilidade.

Note-se que a regra usada na etapa de resolução não tem condições (isto é corresponde a uma regra simples) a query atômica correspondente é simplesmente eliminada.

A etapa da resolução pode diminuir, manter inalterado ou aumentar o número de átomos na query. O processo de resolução termina com sucesso quando não existem átomos no lado esquerdo da query. Cada query atômica foi demonstrada ser uma consequencia das regras. Ele falha quando uma query atômica não pode ser unificada com o átomo cabeça de qualquer regra. Pode ser indeterminado quando alguma seqüência de regras pode ser usada repetidamente sem reduzir o número de query atômicas.

O PROLOG calcula uma query alternando etapas de unificação e resolução substituindo query atomicas por conjunções possivelmente vazias de condições de átomos até que não haja mais query atômicas à esquerda.

Vamos agora seguir o curso do cálculo feito pelo PROLOG. Como exemplo, tomemos a query atômica `caminho2(a,X)` e as regras do nosso programa de grafo dirigido. Há apenas uma regra para o predicado `caminho2`, então a primeira etapa requer uma unificação da query atômica com a cabeça dessa regra:

```
?- caminho2(a,X).
caminho2(S,T) :- aresta(S,U), aresta(U,T).
```

Substituindo `a` por `S` e `T` por `X` em ambos (na query e na regra), fica:

```
?- caminho2(a,T).
caminho2(a,T) :- aresta(a,U), aresta(U,T).
```

O PROLOG agora a query pelas condições da regra tratando a conjunção de átomos como uma query e tomando a primeira regra cuja cabeça é unificável com o primeiro átomo da conjunção.

```
?- aresta(a,U), aresta(U,T).
aresta(a,b).
```

A unificação requer a substituição de `b` por `U` o que dá:

```
?- aresta(a,b), aresta(b,T).
aresta(a,b).
```

Aqui não há condições na regra, assim a query atômica `aresta(a,b)` é simplesmente removida. Segue-se tomando a primeira regra cuja cabeça é unificável com o único átomo remanescente na query:

```
?- aresta(b,T).
aresta(b,d).
```

A unificação, substituindo `d` por `T` nos dá:

```
?- aresta(b,d).
aresta(b,d).
```

Aqui não há condições na regra então a query `aresta(b,d)` é removida deixando uma query vazia o que indica o término com sucesso da computação. Já que `T` foi substituido por `X` e então `d` por `T` a substituição final é `d` por `X` e então o PROLOG pode reportar `X=d`. A substituição de variáveis que não aparecem na query original não são reportadas.

Vale lembrar que a computação falha quando o primeiro átomo da query não pode ser unificado com a cabeça de qualquer regra.

Claramente, para um dado programa não existe um ponto que mereça a aplicação inicial da query, já que a finalização com sucesso do programa requer que o primeiro átomo seja eliminado em algum ponto.

Entretanto, pode ser que em algum ponto prévio da computação a cabeça de mais do que uma regra possa ter sido unificado com o primeiro átomo da query naquele estágio.

 ©88-09, Pedro Kantek

Neste caso, a falha inicial deve causar um chamado *backtracking* para tentar com a outra regra alternativa que foi sobrepujada.

Quando o *backtracking* ocorre todas as substituições que foram feitas a partir do ponto em que houve a divisão, são desfeitas desfazendo-se os efeitos que elas causaram. A lista de átomos na query também é restaurada. A computação retorna a um estágio prévio e tenta novamente com uma nova escolha de regra. As implementações de PROLOG sempre tentam as regras para um determinado predicado na ordem em que elas ocorrem no programa.

EXERCÍCIO 31 Para o mesmo programa conforme o exercício 1, liste as etapas de resolução para as seguintes queries:

1. `caminho2(X,d)`
2. `caminho(b,a)`

O usuário pode provocar um *backtracking* digitando um ponto e vírgula (" ; ") após uma computação bem sucedida. O resto da computação a seguir é como se tivesse ocorrido uma falha neste ponto. Se a computação for bem sucedida a substituição resultante usualmente será diferente daquela obtida previamente refletindo o fato de que existe mais de uma maneira de usar as regras para satisfazer uma dada query. O leitor é encorajado a seguir o *backtracking* que ocorre quando a computação para a query `caminho2(a,X)` para gerar novas substituições para `X`.

EXERCÍCIO 32 Investigue as etapas que ocorrem quando o usuário provoca *backtracking* após a computação para `caminho2(a,X)` ilustrada na sessão 3, notando que nenhuma resposta bem sucedida deve ser achada.

O comando **!** ou *cut* pode ser usado como uma condição em uma regra, também pode ser usado como um átomo em uma query. Seu efeito lógico é simplesmente apresentar sucesso, mas com um efeito colateral que é operacional

> O *backtracking* para tentar outras regras para atomos de query previamente eliminados é evitado.

O uso principal do *cut* é quando uma solução satisfatória para alguma query já foi encontrada e sabe-se que tentativas de buscar outras soluções não são necessárias. Em geral, é aconselhável evitar o uso explícito de cut tanto quanto possível.

A negação por falha pode ser definida usando *cut*. Consideremos as seguintes regras para a definição do predicado `nãoaresta(S,T)`:

```
nãoaresta(S,T) :- aresta(S,T), !, fail.
nãoaresta(S,T).
```

E a query atômica `nãoaresta(c,a)` que leva a `aresta(c,a),!,fail` pela unificação e resolução com a primeira regra acima. Já que `aresta(c,a)` é verdadeira, o `cut` é alcançado e então `fail`. Normalmente `fail` deveria causar back-tracking para tentar a segunda regra para `nãoaresta` mas esta possibilidade foi cortada pelo `cut` logo a query `nãoaresta(c,a)` é forçada a falhar também.

Agora consideremos a query `nãoaresta(a,c)` que leva a `aresta(a,c),!,fail`. Já que `aresta(a,c)` não é verdadeiro, a falha ocorre antes do cut ser alcançado e o backtracking pode tentar uma segunda regra para `nãoaresta`, que tem sucesso incondicional.

Átomos condicionais `...->...` podem ser definidos usando-se cut. Considere a seguinte regra para definir o predicado `salvo(S,T)`:

```
salvo(S,T) :- aresta(S,T) -> aresta(T,S); aresta(S,S).
```

é equivalente a

```
salvo(S,T) :- aresta(S,T), !, aresta(T,S); aresta(S,S).
```

É fácil ver que esta combinação expressa uma escolha condicional, com o resultado dependendo do sucesso ou falha do primeiro átomo.

9.1.6 Dados

Os dados usados para mostrar o PROLOG até agora eram muito simples: um conjunto finito de elementos usados para identificar nodos de um grafo. Vale lembrar que a estrutura do grafo foi representada por fatos específicos dentro do programa, assim não os consideramos realmente como dados. Lembre-se que o mero uso de constantes em predicados causa que elas sejam adicionados aos dados: não existe declaração separada para as constantes.

Mais típico na programação em lógica na prática é o uso de estruturas de dados mais ricas, incluindo listas e árvores. PROLOG provê construtores *built-in* para listas e construtores para outras estruturas de dados, como por exemplo, árvores são introduzidos simplesmente usando-os em argumentos de predicados assim como as constantes. Componentes de listas e dados estruturados são dados sem nenhuma restrição: PROLOG não tem tipos e qualquer disciplina no uso de dados é mantida implícita. Argumentos de predicados podem ser termos arbitrariamente complexos formados a partir de constantes, variáveis e construtores de dados.

A notação de listas finitas em PROLOG é `[D1,...,Dn]`. Se `L` é uma lista, então `[D1,...,Dn|L]` é a lista que se obtem juntando `L` à lista `[D1,...,Dn]`. O predicado `append(L1,L2,L3)` é verdadeiro quando `L3` é a lista formada agregando (*append*) `L2` a `L1`.

`membro(D,L)` é verdadeiro quando `D` ocorre como qualquer componente da lista `L`.

`select(D,L1,L2)` é verdadeiro quando `D` é um componente de `L1` e `L2` é a lista obtida de `L1` pela remoção de uma única ocorrência de `D`.

Cabeças e Caudas de listas podem ser extraídos pela unificação. Vejamos como: Quando `X` e `Y` são variáveis, o termo `[X|Y]` é unificável com qualquer lista não vazia `[D1,...,Dn]` substituindo-se `D1` por `X` e `[D2,...,Dn]` por `Y`. A lista vazia `[]` é unificável apenas com ela mesmo.

EXERCÍCIO 33 Indique quais dos seguintes pares de átomos são unificáveis dando os unificadores quando possível

1. `append([D|L1],L2,[D|L3])` e `append(X,[a,b,c],d)`.
2. `append([D|L1],L2,[D|L3])` e `append([a,b,c],[],X)`.
3. `append(X,[],[a,b])` e `append([a],[b],[a,b])`.

Um bom exemplo de processamento em listas em PROLOG é a definição do predicado `append`, que consiste das seguintes duas regras:

```
append([],L2,L2).
append([D|L1],L2,[D|L3]) :- append(L1,L2,L3).
```

A definição de `membro` abaixo ilustra o uso da variável anônima "_" que é um coringa. Note que a ordem das duas regras é crucial:

```
membro (D,[D|_]).
membro (D,[_|L) :- membro(D,L).
```

O PROLOG trata qualquer variável começando com _ como sendo anônima.

EXERCÍCIO 34 Escreva a regra para o predicado `select(D,L1,L2)`, o qual seja verdadeiro quando `D` seja um componente de `L1` e `L2` é a lista obtida de `L1` pela remoção de uma única ocorrência de `D`.

 ©88-09, Pedro Kantek

Conjuntos são representados em Prolog na forma de listas sem componentes repetidos. Mapeamentos finitos (tais como tabelas de símbolos) são representadas como listas de pares de argumentos. Vários predicados do PROLOG permitem o manuseio destas listas.

A aplicação de um construtor de dados `C` aos argumentos `D1,...Dn` é escrita simplesmente como o termo `C(D1,...,Dn)`. Quando `X1,...,Xn` são variáveis, o termo `C(X1,...,Xn)` é unificável com qualquer aplicação do mesmo construtor de dados `C` ao mesmo número de componentes. De maneira mais geral, dois termos construidos por construtores de dados são unificáveis se e somente se ambos são aplicações do mesmo construtor de dados e os pares de seus correspondentes argumentos são todos unificáveis.

Os operadores aritméticos são essencialmente construtores de dados eles próprios. Note que se `N1` e `N2` são inteiros, o termo `1+2*` não é um inteiro e sim uma estrutura de dados, embora possa ser unificada com um inteiro a seguir. Existe ainda o predicado `true` que é sempre verdadeiro e `fail` que é sempre falso. A igualdade em PROLOG é `=:=`.

EXERCÍCIO 35 Forneça ao PROLOG regras para obter o predicado `Ack` que calcule a função de Ackerman´s definidas pelas equações:

$$Ack(0, y) = y + 1$$

$$Ack(x, 0) = Ack(x - 1, 1), \forall x > 0$$

$$Ack(x, y) = Ack(x - 1, Ack(x, y - 1)), \forall x, y > 0$$

9.2 Um tutorial

Na hora de escrever comandos em PROLOG, deve-se ter em mente a seguinte tabela

português	lógica de predicados	prolog
E	$\wedge$	,
OU	$\vee$	;
somente se	$\leftarrow$	:-
não	$\neg$	não

Variáveis são escritas em maíusculo. Constantes em minúsculo. A variável *underline* é considerada uma variável de trabalho, não nomeada. Acompanhe nos exemplos

9.3 Lógica de Primeira Ordem

O mundo está constituído de **objetos**. Estes têm **propriedades**, que os distinguem de outros objetos. Entre os objetos existem **relações**, algumas das quais são **funções**[1]. Seja por exemplo

Objetos alunos, carteiras, sanduiches, questões de prova, notas, leis, deputados, cores, times campeões...

Relações maior que, menor que, dentro de, da cor tal, próximo a, mais difícil que, irmão de...

propriedades verde, redondo, gordo, difícil, fácil, breve, chato, ...

funções dono de, pai de, o dobro de,

[1] Uma função é um tipo especial de relação que tem apenas um resultado

A LPO é boa para descrever aspectos particulares do mundo. Muita coisa fica de fora (categorias, tempo, acontecimentos, ...), mas longe de ser uma deficiência esta característica é vantagem. Ela permite modelar o mundo que queremos representar com certa liberdade, facilitando a tarefa de modelagem. Por exemplo, professor pode ser uma propriedade de pessoas, ou uma relação de turma e matéria, ou uma função de turma-homenageia (supondo que uma turma só possa homenagear um professor...).

9.3.1 Síntaxe

Usar-se-á a forma de Backus-Naur para a descrição da síntaxe da LPO

- Oração → Oração-atômica | Oração Conectivo Oração | Quantificador Variável Oração | ¬ Oração | (Oração)
- Oração-atômica → Predicado(Termo) | Termo
- Termo → Função(Termo) | Constante | Variável
- Conector → ⇒ | ∧ | ∨ | ⟺
- Quantificador → ∀ | ∃
- Constante → A | X_1 | Paulo...
- Variável → a | x | s | ...
- Predicado → Antes | Temcor | Chovendo ...
- Função → Dobro | Pai-de ...

Constantes O nome identifica um objeto do mundo, se bem que nem todos os objetos precisem ter nome.

Predicados Cada predicado se aplica a um determinado número de objetos. Assim, o predicado *Irmão* associa 2 objetos. Um predicado, portanto, atua sobre uma *t-upla*. Uma t-upla é um conjunto de número fixo de objetos em ordem determinada.

Funções Algumas relações são também funções. Ou seja existe uma relação que gera um único resultado. Pode ser mais fácil implementar a função (por exemplo, Coseno) do que introduzir toda a tabela de cosenos. No modelo, a função associa uma t-upla de $n+1$ elementos, onde o último é o resultado da aplicação da função aos n primeiros elementos.

Termos É uma expressão lógica que se refere a um objeto. As constantes são termos.

Orações atômicas Uma oração atômica está formada por um predicado e uma lista de termos entre parênteses. Por exemplo Irmão(Carlos,José) diz que carlos é irmão de josé. Pode ser complexa, como Casado(Pai(Carlos),Mãe(José))

 ©88-09, Pedro Kantek

Orações complexas Usando-se os conectores lógicos podem-se construir orações complexas Exemplos:

Irmão(Carlos,José) $\wedge$ Irmão(Carlos,Luis)	é verdadeira quando ambas o são
Maior(Carlos,30) $\vee$ Menor(Carlos,30)	é verdadeira se uma delas o for
Maior(Carlos,30) $\Rightarrow$ $\neg$Menor(Carlos,30)	se carlos tem mais de 30 anos, então ele não tem menos que 30 anos
$\neg$Irmão(Alfredo,João)	é verdadeiro se alfredo não é irmão de joão

Quantificadores Usadas para estabelecer propriedades de grupos completos de objetos em vez de enumerá-los pelos seus nomes. São dois: o universal ($\forall$) e o existencial ($\exists$).

Para expressar o fato de que todos os gatos são mamíferos, empregam-se dois predicados unários, a saber *Gato* e *Mamífero*. Assim o fato de que Binho é um gato, se expressa *Gato(Binho)* e *Mamífero(Binho)*. A seguir a frase que informa que quem é gato é também mamífero, se escreve como $\forall$ x Gato(x) $\Rightarrow$ Mamífero(x)

O nome *universal* vem do fato que a assertiva que segue o símbolo é verdadeira para todos os objetos do universo. Assim, escrever $\forall$ x P, equivaleria a fazer a conjunção (usando $\wedge$) de todos os objetos do universo substituindo x. Supondo os objetos A, B, C, ... levaria a P(A) $\wedge$ P(B) $\wedge$ P(C)...

As variáveis serão representadas em minúsculo. As constantes, predicados e funções em maiúsculo. Quando um termo não tem variáveis é conhecido como termo de base.

Lembre-se que a conseqüência de $\forall$ x Gato(x) $\Rightarrow$ Mamífero(x) pode ser Gato(Carlos) $\Rightarrow$ Mamífero(Carlos). Esta frase está correta e é verdadeira. Olhando a definição de $\Rightarrow$, verifica-se que a premissa é falsa nada se pode falar sobre a conseqüência.

Já o quantificador existencial serve para fazer afirmações sobre algum objeto. Assim, para afirmar que Binho tem um irmão que é também um gato, escreve-se

$\exists$ x irmão(x,Binho) $\wedge$ Gato(x)

Ao escrever $\exists$ x P afirma-se que P é verdadeiro para algum objeto do universo. Pelo que, essa expressão pode ser substituída pela disjunção (usando $\vee$) de todos os objetos do universo. Supondo que sejam A, B, C... ter-se-ia P(A) $\vee$ P(B) $\vee$ P(C)... e esta expressão seria verdadeira se o fosse para pelo menos um dos objetos. Uma particularidade de $\exists$ ocorre quando se deseja afirmar que existe um e um só. Usa-se o símbolo$\exists$!.

Os quantificadores podem ser usados juntos, veja-se nos exemplos

$\forall$ x,y Pai(x,y) $\Rightarrow$ Filho(y,x)	Se x é pai de y então y é filho de x
$\forall$ x,y Irmão(x,y) $\Rightarrow$ Irmão(y,x)	Se x é irmão de y então y é irmão de x
$\forall$x, $\exists$ y Amores(x,y)	Todas as pessoas amam alguém
$\exists$ y, $\forall$ x Amores(x,y)	Sempre há alguém a quem todos amam
$\forall$ x $\neg$ Gostade(x,bucho) equivale a $\neg\exists$ x Gostade(x,bucho)	Todos destestam bucho equivale a não há quem goste de bucho
$\forall$ x Gostade(x,sorvete) equivale a $\neg\exists$ x $\neg$Gostade(x,sorvete)	Todos gostam de sorvete equivale a não há quem não goste de sorvete

Já que $\forall$ é uma conjunção do universo e $\exists$ a sua disjunção não é surpresa que elas obedeçam à Lei de De Morgan, como se pode ver

$\forall x \neg P \equiv \neg \exists\ x\ P \quad \neg P \wedge \neg Q \equiv \neg(P \vee Q)$
$\neg \forall x\ P \equiv \exists\ x \neg P \quad \neg (P \wedge Q) \equiv \neg P \vee \neg Q$
$\forall\ x\ P \equiv \neg \forall x \neg P \quad P \wedge Q \equiv \neg(\neg P \vee \neg Q)$
$\exists\ x\ P \equiv \neg \forall\ x \neg P \quad P \vee Q \equiv \neg(\neg P \wedge \neg Q)$

Igualdade A igualdade = é usada para afirmar que certos termos se aplicam a um objeto determinado. Por exemplo, Pai(João)=José está afirmando que o pai de Joâo é o José.

Lógicas de ordem superior Na lógica de primeira ordem quantificam-se objetos (que são as entidades de primeira ordem a constituir o universo), mas não se podem quantificar as relações ou funções que existem entre os objetos. Nas lógicas de ordem superior é possivel quantificar relações e funções. Estas lógicas tem maior capacidade de expressão, mas elas ainda não são bem conhecidas e por isso menos usadas.

O operador λ Para escrever funções e predicados sem lhes atribuir um nome, usa-se o operador λ. Assim se quisermos definir a função que eleva a quarta potencia um número x, podemos fazer

Quarta(x) $\Rightarrow (x^4)$, que aplicada a 3, dará como resposta 81.

Sem usar nomes, pode-se escrever

(λ x x^4) que aplicado a 3, também dará 81. [2]

Nas linguagens Em PROLOG, as variáveis são escritas em maiúsculo e as constantes em minúsculo. As implicações também são invertidas. $Q :- P$ ao invés de $P \Rightarrow Q$. A vírgula separa argumentos e serve também como conjunção. O ponto encerra uma oração.

Em LISP não há diferença entre maiúsculos e minúsculos, pelo que variáveis, em geral, começam com uma interrogação, como no exemplo

```
(todo ?x (implica (y (peludo ?x) (mia ?x) (temgarras ?x)) (gato ?x)))
```

A ordem dos quantificadores é importante e não se deve usar nenhuma variável que não tenha sido previamente quantificada.

Usos de uma LPO

Na representação do conhecimento um **domínio** é um fragmento do mundo sobre o qual se deseja expressar determinado conhecimento. Seja a seguir um exemplo no domínio do parentesco. Os objetos neste domínio são *pessoas.*As propriedades são o gênero e as relações que guardam entre si. Assim, haverá um predicado unário (*homem* e *mulher*) e diversos predicados binários: *progenitor, filho, irmão, irmã, fraterno, prole, filho, filha, esposo, esposa, cônjuge, avô, neto, primo, tio e tia.* Para pai e mãe, usar-se-á uma função, pois do ponto de vista biológicos, todos contam com um de cada. Eis as regras:

- A mãe de alguém é o seu progenitor feminino
  $\forall$ *m,x Mãe(x) = m* $\iff$ *Mulher(m)* $\wedge$ *Progenitor(m,x)*
- O esposo de alguém é o seu cônjuge masculino
  $\forall$ *h,x Esposo(x,h)* $\iff$ *Homem(h)* $\wedge$ *Cônjuge(h,x)*

[2] Na linguagem LISP existe o operador lambda que funciona exatamente assim

 ©88-09, Pedro Kantek

- Masculino e feminino são categorias disjuntas
  $\forall\ x\ Homem(x) \iff \neg Mulher(x)$

- Progenitor e Prole são relações inversas
  $\forall\ p,f\ Progenitor(f,p) \iff Prole(p,f)$

- Um avô é o progenitor do progenitor de alguém
  $\forall\ Avo(n,a) \iff \exists\ p\ Progenitor(n,p) \wedge Progenitor(p,a)$

- Um fraterno é outro filho dos progenitores de alguém
  $\forall\ x,y\ Fraterno(x,y) \iff x \neq y \wedge \exists\ p\ Progenitor(x,p) \wedge Progenitor(y,p)$

Os matemáticos criam axiomas para capturar os feitos básicos sobre um domínio. Definem outras entidades em função desses axiomas e finalmente usam axiomas e definições para demonstrar teoremas.

A pergunta aqui é como saber se foram definidos suficientes axiomas para especificar completamente o domínio ? Uma maneira é estabelecer um conjunto básico de predicados em função dos quais se possam definir todos os outros. No domínio do parentesco, por exemplo, os candidatos a conjunto básico são Prole, Cônjuge, Homem e Mulher.

O problema oposto é ter excesso de axiomas. Na matemática, um axioma independente é aquele que não pode ser obtido a partir de outros. Os matemáticos se esforçam para produzir o conjunto mínimo de axiomas independentes entre si. Já na IA é comum trabalhar com axiomas redundantes, já que eles podem implicar em maior eficiência na demonstração de teoremas.

Outro exemplo a seguir, no domínio dos conjuntos. Aqui precisa-se a capacidade de representar conjuntos, inclusive o vazio. Conjunto serão construídos adicionando um elemento a um conjunto, mediante a união. Precisa-se saber se um elemento é membro de um conjunto e diferenciar conjuntos de outros objetos que não são conjuntos. Usar-se-á o vocabulário comum da teoria de conjuntos: Conjuntovazio é uma constante; Membro e Subconjunto são predicados; Interseção, União, Incorpora são funções. Conjunto é um predicado que se aplica aos conjuntos. Seguem-se 8 axiomas

1. Os únicos conjuntos são o vazio e aqueles que resultam da incorporação de alguma coisa a um conjunto
   $\forall\ c\ Conjunto(c) \iff (c=Conjuntovazio) \vee (\exists\ x,c_2\ Conjunto(c_2) \wedge c=Incorpora(x,c_2))$

2. O conjunto vazio é aquele que não tem incorporado nenhum elemento, ou seja não é possível decompor o conjunto vazio em conjunto menor nem em um elemento.
   $\neg\exists\ x,c\ Incorpora(x,c)=Conjuntovazio$

3. A incorporação de um elemento que já esteja em um conjunto não produz nenhum efeito
   $\forall\ x,c\ Membro(x,c) \iff c=Incorpora(x,c)$

4. Os únicos membros de um conjunto são aqueles que foram incorporados a este conjunto
   $\forall\ x,c\ Membro(x,c) \iff \exists\ y,d\ (c=Incorpora(y,d) \wedge (x=y \vee Membro(x,d)))$

5. Um conjunto é subconjunto de outro se e somente se todos os elementos do primeiro são também do segundo
   $\forall\ c_1,\ c_2\ Subconjunto(c_1,c_2) \iff (\forall\ x\ Membro(x,c_1) \Rightarrow Membro(x,c_2))$

6. Dois conjuntos são iguais se e somente se cada um é subconjunto do outro
   $\forall\ c_1,\ c_2\ (c_1 = c_2) \iff (Subconjunto(c_1,c_2) \wedge Subconjunto(c_2,c_1))$

7. Um objeto é membro da intersecção de dois conjuntos se e somente se ele é membro de cada um dos conjuntos
   $\forall\ x,c_1,c_2\ \ Membro(x,Intersecção(c_1,c_2)) \iff Membro(x,c_1) \wedge Membro(x,c_2)$

8. Um objeto é membro da união de dois conjuntos se e somente se ele é membro de um ou de outro dos conjuntos
   $\forall\ x,c_1,c_2\ \ Membro(x,União(c_1,c_2)) \iff Membro(x,c_1) \vee Membro(x,c_2)$

9.3.2 Mais exemplos

Material baseado em "Cálculo de Predicados e a linguagem PROLOG"de Maria Carolina Monard, ICMC,USP.

1. Alguns gatos sabem caçar, outros não sabem
   $\exists\ g\ Sabecaçar(g) \wedge \exists\ h \neg\ sabecaçar(h)$
   Note que são necessárias duas variáveis g e h pois se fosse usada a mesma, estaria se afirmando que um gato sabe e não sabe caçar.

2. Se Maria não é bonita, nenhuma das mulheres é bonita
   $\neg\ Bonita(Maria) \Rightarrow \forall\ x \neg\ Bonita(x)$

3. João e Luiz viram a orquestra, mas houve quem não a visse
   $Viuorquestra(João) \wedge Viuorquestra(Luiz) \wedge \exists\ x \neg\ Viuorquestra(x)$

4. Nem tudo que reluz é ouro
   $\exists\ x\ (Reluz(x) \wedge \neg\ Ouro(x))$ ou $\neg \forall\ x\ (Reluz\ (x) \Rightarrow Ouro(x))$

9.3.3 Exercícios do [Rus96]

1. Represente as orações seguintes em LPO usando um vocabulário congruente, que você terá que construir:

   (a) Nem todos os estudantes cursam álgebra e estrutura de dados
   $\exists\ x\ (Cursaalgebra(x) \wedge \neg\ Cursaed(x)) \vee (\neg\ Cursaalgebra(x) \wedge Cursared(x))$

   (b) Apenas um estudante reprovou em compiladores
   $\exists\ !\ x\ Reprovou(x,Compiladores)$

   (c) Apenas um estudante reprovou em compiladores e em análise
   $\exists\ !\ x\ (Reprovou(x,Compiladores) \wedge Reprovou(x,Análise))$

   (d) A melhor das notas obtidas em compiladores foi melhor do que a melhor das notas obtidas em análise
   $\exists\ x \exists\ y\ Melhornota(compiladores,x),\ Melhornota(análise,y) \wedge x>y$

   (e) Toda pessoa que não gosta de bucho é inteligente
   $\forall\ x \neg\ Gostabucho(x) \Rightarrow Inteligente(x)$

   (f) Há uma mulher que é paquerada por todos os homens que não gostam de bucho
   $\forall\ h \exists\ m \neg\ Gostabucho(h) \wedge Homem(h) \wedge Mulher(m) \Rightarrow Paquera(h,m)$

   (g) Há uma mulher que paquera a todos os homens que não gostam de bucho
   $\exists\ m \forall\ h\ Mulher(m) \wedge Homem(h) \wedge \neg\ Gostabucho(h) \Rightarrow Paquera(m,h)$

   (h) Há um barbeiro que barbea (corta a barba de) todos os homens do povoado que não se barbeam eles mesmos (Paradoxo de Russel)
   $\exists\ b \forall\ h\ Barbea(b,h) \wedge \neg\ Barbea(h,h)$

   (i) Ninguém gosta de professores que não sejam inteligentes
   $\neg \forall\ x \forall\ p\ Professor(p) \wedge \neg\ Inteligente(p) \Rightarrow Gosta(x,p)$

 ©88-09, Pedro Kantek

(j) Os políticos podem enganar a alguns sempre, enganar a todos às vezes, mas não podem enganar a todos sempre.
$\forall$ *x Politico(x)* $\wedge$ *(Enganar(p,Alguns,Sempre)* $\vee$ *Enganar(p,Todos,Àsvezes)* $\vee\neg$ *Enganar(p,Todos,Sempre)*

2. Represente a oração "Todos os espanhois falam os mesmos idiomas"mediante o cálculo de predicados. Utilize Fala(x,i) significando que a pessoa x fala o idioma i.

3. Escreva os axiomas que descrevam o predicado Neto, Bisavô, Irmão, Irmã, Filha, Filho, Tio, Tia, Cunhado e Primo, usando como primitivos Homem, Mulher, Cônjuge, Progenitor.

Desafio

- Resolver os exercícios postos nesta aula, formulando os problemas em lógica de primeira ordem.

9.4 Inferência na LPO

As regras de inferência começaram a ser vistas no *cálculo proposicional* e eram

Modus Ponens Se $\alpha \Rightarrow \beta$ e se α é verdadeira, daí decorre que β é verdadeira também. Por exemplo, diante do fato de que "todos os gatos miam"ao observar o gato Binho, pode-se inferir que ele mia.

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

Eliminação de $\wedge$ Se existe uma conjunção, todos os componentes são verdadeiros. Por exemplo, se a conjunção "hoje é segunda"$\wedge$ "estamos em dezembro"$\wedge$ "faz calor"for verdadeira, então cada uma dos fatores o será.

$$\frac{\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n}{\alpha_i}$$

Introdução de $\wedge$ Se existe um conjunto de orações verdadeiras, é possível inferir sua conjunção. Seria o processo oposto ao do exemplo anterior.

$$\frac{\alpha_1, \alpha_2, ..., \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n}$$

Introdução de $\vee$ Dada uma oração, é possível inferir sua disjunção com todas as demais orações. Se a oração dada é verdadeira, verdadeira será a expressão dessa oração ligada com $\vee$ a todas as demais orações.

$$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee ... \vee \alpha_n}$$

Eliminação da dupla negação Uma oração duplamente negada é verdadeira. *Exemplo real: Uma vez vi um estagiário escrevendo em um programa "Não tecle F10 para não atualizar"*

$$\frac{\sim\sim \alpha}{\alpha}$$

Resolução Unitária Dada uma disjunção verdadeira, se um dos disjuntos é falso, pode-se inferir que o outro é verdadeiro.

$$\frac{\alpha \vee \beta, \quad \sim \beta}{\alpha}$$

Resolução Já que β não pode ser ao mesmo tempo, verdadeira e falsa, um dos outros disjuntos deve ser uma das premissas. Em outras palavras, a implicação é transitiva.

$$\frac{\alpha \vee \beta, \quad \sim \beta \vee \gamma}{\alpha \vee \gamma}$$

ou seu equivalente

$$\frac{\sim \alpha \Rightarrow \beta, \quad \beta \Rightarrow \gamma}{\sim \alpha \Rightarrow \gamma}$$

Além destas, são necessárias outras 3 regras de inferência para manusear as orações em LPO com quantificadores. Vão se produzir substituições de variáveis por entes particulares. Escrever-se-á $SUBST(\theta, \alpha)$ para representar o resultado obtido de aplicar a substituição θ na oração α. Por exemplo, $SUBST(\{x/Maria, y/Carlos\}, Gosta(x, y)) = Gosta(Maria, Carlos)$. Eis as novas regras de inferência

Eliminação universal Para toda oração α, variável v e termo de base g[3]

$$\frac{\forall v \quad \alpha}{SUBST(\{v/g\}, \alpha)}$$

Por exemplo em $\forall x Ama(x, Gatos)$ pode-se utilizar a substituição $\{x/Luiz\}$ e inferir que $Ama(Luiz, Gatos)$

Eliminação existencial Para toda oração α, variável v, e símbolo constante k que não apareça em nenhuma parte da Base de Conhecimento,

$$\frac{\exists\, v \quad \alpha}{SUBST(\{v/k\},\ \alpha)}$$

Por exemplo em $\exists\, x Beber(x, Tequila)$ pode-se inferir que $Beber(Juanito, Tequila)$ desde que $Juanito$ não apareça na base de conhecimentos.

Introdução existencial Para qualquer oração α, variável v que não esteja em α e termo de base g que não esteja em α,

$$\frac{\alpha}{\exists\, v \quad SUBST(\{g/v\},\ \alpha)}$$

Por exemplo, em $Ama(Carlos, Sofia)$ pode-se inferir que $\exists\ x\ Ama(x, Sofia)$.

É importante que a constante empregada para substituir a variável nas regras de eliminação existencial seja uma variável nova. A não obediência a este requisito gera conseqüências ilógicas.

Por exemplo, na oração $\exists\, x\ Pai(x, Joaquim)$ (Joaquim tem um pai...) ao substituir a variável x por Joaquim, fica $Pai(Joaquim, Joaquim)$ o que não é conseqüência da oração original.

9.4.1 Exemplo de uma demonstração

Seja a seguinte situação

> A lei estabelece que se considera um delito que um cidadão americano venda armas a nações inimigas. O país Nono, inimigo dos Estados Unidos tem alguns projéteis, todos vendidos pelo Coronel West, um americano.

[3]Termo de base é aquele em que não há variáveis. Há uma constante ou uma função aplicada a alguns termos de base

 ©88-09, Pedro Kantek

Desejamos demonstrar que West é um delinqüente. Representando estes fatos em LPO

- a) É delito que um americano venda armas a nações inimigas
  $\forall x, y, z, Americano(x) \wedge Arma(y) \wedge Nacao(z) \wedge Hostil(z) \wedge Vende(x, y, z) \Rightarrow Delinquente(x)$
- b) Nono tem alguns projéteis
  $\exists\, x\ Possui(Nono, x) \wedge Projeteis(x)$
- c) Todos os projéteis de Nono foram vendidos pelo Coronel West
  $\forall x\ Possui(Nono, x) \wedge Projeteis(x) \Rightarrow Vende(West, Nono, x)$
- d) Projéteis são armas
  $\forall x\ Projeteis(x) \Rightarrow Arma(x)$
- e) Os inimigos dos Estados Unidos são hostís
  $\forall x\ Inimigo(x, EUA) \Rightarrow Hostil(x)$
- f) West que é americano
  $Americano(West)$
- g) O país Nono
  $Nacao(Nono)$
- h) Nono inimigo dos Estados Unidos
  $Inimigo(Nono, America)$
- i) América é uma nação
  $Nacao(America)$

A demonstração consiste em uma série de aplicações das regras de inferência

- j) De (b) e pela eliminação existencial
  Possui(Nono,M1) $\wedge$ *Projétil(M1)*
- De (j) e pela eliminação E
  k) *Possui(Nono,M1)* e
  l) *Projétil(M1)*
- m) De (d) e mediante a eliminação universal
  Projetil(M1) $\Rightarrow$ *Arma(M1)*
- n) De (l) e (m) com Modus Ponens
  Arma(M1)
- o) De (c) e pela Eliminação Universal
  Possui(Nono,M1) $\wedge$ *Projétil(M1)* $\Rightarrow$ *Vende(West,Nono,M1)*
- p) De (o) e (j) e pelo Modus Ponens
  Vende(West,Nono,M1)
- q) De (a) e pela Eliminação Universal aplicada 3 vezes
  Americano(West) $\wedge$ *Arma(M1)* $\wedge$ *Nação(Nono)* $\wedge$ *Hostil(Nono)* $\wedge$ *Vende(West,Nono,M1)*
  $\Rightarrow$ *Criminoso(West)*
- r) De (e) e pela Eliminação Universal
  Inimigo(Nono,América) $\Rightarrow$ *Hostil(Nono)*
- s) De (h), (r) e Modus Ponens
  Hostil(Nono)

- t) De (f), (g), (n), (p), (s) e Introdução E
  Americando(West) $\wedge$ *Arma(M1)* $\wedge$ *Nação(Nono)* $\wedge$ *Hostil(Nono)* $\wedge$ *Vende(West,Nono,M1)*

- u) De (q), (t) e Modus Ponens
  Criminoso(West)

Se o processo de encontrar a demonstração é definido como um processo de busca, então encontrar a solução é achar o estado meta que a contém. Olhando este problema como um de busca, ter-se-ia

Estado Inicial A Base de conhecimento formada pelas orações (a) até (i)

Operadores As regras de inferência estudadas

Meta Uma BC que contenha a oração Criminoso(West)

Este exemplo mostra algumas características importantes:

- A demonstração consta de 14 passos

- O fator de ramificação aumenta conforme cresce a BC, já que em muitas regras de inferência se misturam fatos novos com antigos.

- A eliminação universal sozinha introduz um grande fator de crescimento já que a variável se pode substituir por qualquer termo de base.

- Gasta-se muito combinando orações atômicas em conjunções concretizando regras universais para assim efetuar cotejos e depois aplicar o Modus Ponens.

Uma maneira de abreviar estas etapas é usar a regra de Modus Ponens Generalizada: Para as orações atômicas p_i, p'_i e q nas quais existe uma substituição θ na qual $SUBST(\theta, p_i)$ para quaisl i,

$$\frac{p'_1,\ p'_2,\ ...,\ p'_n,\ (p_1 \wedge p_2 \wedge ... \wedge p_n \Rightarrow q)}{SUBST(\theta, q)}$$

Nesta regra, há $n+1$ premissas: As n orações atômicas p'_i e uma implicação. Há uma conclusão: o resultado obtido mediante a aplicação da substituição na q consequente. Por exemplo, no caso de West e o projétil

p_1 é Projétil(x)	p'_1 é Projétil(M1)
p_2 é Possui(Nono,x)	p'_2 é Possui(y,M1)
θ\{x / M1, y/Nono\}	q é Vende(West,Nono,x)
$SUBST(\theta, q)$ é Vende(West,Nono,M1)	

O Modus Ponens Generalizado é eficiente pelas três razões seguintes

1. Seus passos são mais globais ao combinar várias inferências pequenas em uma só

2. Os passos que realiza são sensatos: utiliza substituições que garantem a sua utilidade em vez de experimentar randomicamente a eliminação universal. O algoritmo da unificação utiliza duas orações e produz uma substituição pela qual as orações anteriores resultam idênticas sempre que tal substituição exista.

3. Precisa que todas as orações da BC estejam na forma canônica. Exigir isto acelera o processamento pois evita a conversão a cada passo.

 ©88-09, Pedro Kantek

Forma canônica

O esforço agora é construir um mecanismo de inferência junto com a regra de inferência: a versão generalizada do Modus Ponens. Isto exige que todas as orações da BC em uma forma tal que coincida com as premissas da regra de Modus Ponens. Isto determina que cada oração da BC seja uma oração atômica ou uma implicação com uma conjunção de orações atômicas no lado esquerdo e apenas um átomo no lado da direita. A este tipo de orações, denomina-se **Orações de Horn**. À BC constituída apenas de orações de Horn chama-se **Formal normal de Horn**.

Cláusulas de Horn Um tipo especial de cláusula que desempenha papel importante na IA e mesmo em outras partes da ciência da computação. Elas foram investigadas inicialmente pelo lógico Alfred Horn na década de 50. Existem 3 tipos de cláusulas de Horn

- Um simples átomo (também chamado de "fato")
- Uma implicação (chamada regra), cujo antecedente seja uma conjunção de literais positivos e cujo conseqüente seja um único literal positivo
- Um conjunto de literais negativos, escritos em uma implicação na qual o antecedente seja uma conjunção de literais positivos e a conseqüente seja vazia. Esta forma é chamada objetivo (goal).

Nas cláusulas de Horn, não há disjunções de literais positivos.

A conversão de orações em orações de Horn é feita quando as orações são introduzidas pela primeira vez na BC. Usa-se a eliminação existencial e a introdução de $\wedge$. Por exemplo, a expressão $\exists\ x\ Possui(Nono, x) \wedge Projetil(x)$ é convertida em duas orações atômicas de Horn: $Possui(Nono, M1)$ e $Projetil(M1)$. Depois que todos os quantificadores existenciais foram eliminados, podem ser excluídos também os quantificadores universais de modo que $\forall y\ Possui(y, M1)$já se escreva como $Possui(y, M1)$. Isto é apenas uma abreviação, y continua sendo uma variável quaintificada universalmente, mas é mais fácil ler e escrever orações sem os quantificadores.

Unificação

O algoritmo $UNIFICAR$ pega duas orações p e q realiza substituições permitidas de maneira que p e q fiquem iguais. Caso não exista tal unificação o algoritmo produzirá um erro. Formalmente

$UNIFICAR(p, q) = \theta$ onde $SUBST(\theta, p) = SUBST(\theta, q)$

θ é conhecido como UNIFICADOR das duas orações. Veja-se o exemplo

$$Conhece(Joaquim, x) \Rightarrow Odeia(Joaquim, x)$$

Que pode ser traduzido como Joaquim odeia a todas as pessoas que conhece. Queremos usar esta oração junto com Modus Ponens para saber quem são as pessoas que o Joaquim odeia. Ou seja, precisa-se saber quais orações da BC se unificam com $Conhece(Joaquim, x)$ e depois aplicar este unificador a $Odeia(Joaquim, x)$. Supondo que a BC contém as orações seguintes

$Conhece(Joaquim, Jane)$
$Conhece(y, Leonidas)$
$Conhece(y, Mae(y))$
$Conhece(x, Isabel)$

Lembrando que x e y estão universalmente quantificadas. Ao unificar a oração acima com dada uma das componentes da BC fica:
$UNIFICAR(Conhece(Joaquim, x), Conhece(Joaquim, Jane)) = \{x/Jane\}$
$UNIFICAR(Conhece(Joaquim, x), Conhece(y/Leonidas)) = \{x/Leonidas, y/Joaquim\}$
$UNIFICAR(Conhece(Joaquim, x), Conhece(y, Mae(y))) = \{y/Joaquim, x/Mae(Joaquim)\}$
$UNIFICAR(Conhece(Joaquim, x), Conhece(x, Isabel)) = erro$

Este último caso dá erro porque x não pode ser Joaquim e Isabel ao mesmo tempo. No entanto, há uma incongruência, pois uma frase afirma que todos conhecem Isabel, e a outra que o Joaquim odeia a todos aos quais conhece, logo, Joaquim deveria odiar Isabel.

A maneira de corrigir este defeito é normalizar os duas orações que vão ser unificadas mudando os nomes delas a fim de evitar a repetição de variáveis. Fazendo isto, a subsituição poderia ser
$UNIFICAR(Conhece(Joaquim, x_1), Conhece(x_2, Isabel)) = \{x_1/Isabel, x_2/Joaquim\}$.

Retornando à demonstração anterior e usando o Modus Ponens Generalizado e as formas de Horn, fica

- a) Convertendo todas as orações para a forma de Horn, fica:
  Americano(x) $\wedge$ *Arma(y)* $\wedge$ *Nação(z)* $\wedge$ *Hostil(z)* $\wedge$ *Vende(x,z,y)* $\Rightarrow$ *Criminoso(x)*
- b) *Possui(Nono,M1)*
- c) *Projétil(M1)*
- d) *Possui(Nono,x)* $\wedge$ *Projétil(x)* $\Rightarrow$ *Vende(West,Nono,x)*
- e) *Projétil(x)* $\Rightarrow$ *Arma(x)*
- f) *Inimigo(x,America)* $\Rightarrow$ *Hostil(x)*
- g) *Americano(West)*
- h) *Nação(Nono)*
- i) *Inimigo(Nono,America)*
- j) *Nação(America)*
- k) De (c) e (e) através de Modus Ponens, fica
  Arma(M1)
- l) De (i) e (f) com Modus Ponens, fica
  Hostil(Nono)
- m) De (b), (c) e (d) com Modus Ponens, fica
  Vende(West,Nono,M1)
- n) De (g), (k), (h), (l), (m), (a) e Modus Ponens fica
  Criminoso(West)

A regra de Modus Ponens Generalizada (MPG) pode ser usada de duas maneiras:

1. Começar pelas orações que estão na BC gerando novas conclusões. Estas também podem ser usadas para novas inferências. Este método é chamado **encadeamento à frente** e é usado quando um novo conhecimento é agregado à BC e deseja-se saber as suas conseqeuncias.
2. Começar com o fato que se deseja demonstrar. Procurar as orações que permitem chegar a essa conclusão e finalmente localizar as premissas correspondentes. O nome aqui é **encadeamento para trás**, já que se usa o Modus Ponens em sentido inverso. É usado quando é necessário demonstrar uma meta.

 ©88-09, Pedro Kantek

9.4.2 Resolução × refutação

A demonstração de um teorema pode seguir dois caminhos. O primeiro, chamado RESOLUÇÃO, vai operando os axiomas e os teoremas até topar com o teorema que se quer provar. O segundo chamado REFUTAÇÃO, parte da negação lógica do que se quer provar e mediante as tratativas usuais chega a um ponto em que uma contradição é encontrada. Está provado o teorema original.

Este caminho não é novo. A escola pitagórica há mais de 2400 anos já a utilizava. Denominava-a de *reductio ad absurdum* ou em português, redução ao absurdo. Esta estratégia de demonstração, sempre pressupõe verdadeiro o oposto do que se quer criar. Procedendo rigorosamente nas manipulações lógicas, caso se chegue a um absurdo, está provada a tese original.

Vejamos como Howard Eves[4] descreve a descoberta da irracionalidade de $\sqrt{2}$. Antes de mostrar o teorema, vale a referência de que esta descoberta foi uma bomba no seio dos pitagóricos. Eles – até então – julgavam que todo o universo poderia ser explicado em termos de números inteiros ou seus aparentados os números racionais. Tendo descoberto o primeiro irracional e com receio de descobrir outros os pitagóricos tramaram entre sí que ninguém contaria para estranhos à comunidade esta descoberta. Conta a lenda que Hipaso (um pitagórico) teria dado com a língua nos dentes. Sua punição foi ser lançado ao mar. Outra versão diz que ele foi expulso, tendo lhe sido erigido um túmulo, como se morto estivesse.

Antes de provar que $\sqrt{2}$ é irracional, observemos que se x é um inteiro positivo, então x^2 é par se e somente se x é par.

Na tentativa de mostrar que $\sqrt{2}$ é irracional por redução ao absurdo, comecemos supondo que $\sqrt{2}$ é racional. Ou seja $\sqrt{2} = a/b$ em que a e b são inteiros e primos entre sí. Se eles não forem primos entre sí, deve-se simplificar a fração, até a situação em que não haja mais simplificações possíveis: agora eles são primos entre si. Agoram, pode-se escrever

$$a = b\sqrt{2}$$

Elevando esta expressão ao quadrado, fica

$$a^2 = 2b^2$$

Como a^2 é o dobro de um inteiro, concluímod que a^2 é par, logo pelo axioma acima declarado, a também é par. Disto, faz-se $a = 2c$ e reescreve-se a última equação:

$$4c^2 = 2b^2$$

ou simplificando

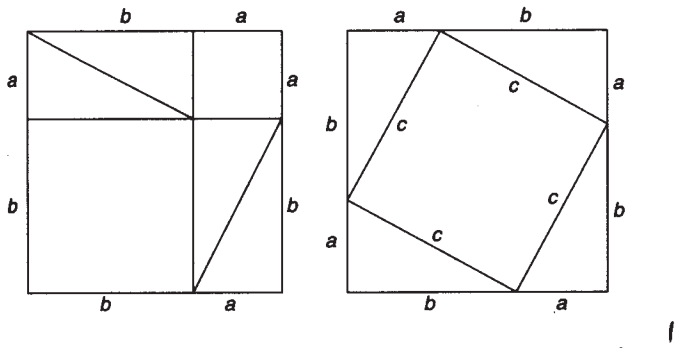
$$2c^2 = b^2$$

Agora, pode-se concluir que b^2 é par e portanto (pelo axioma), b também é par. Mas, isto é impossível, pois admitiu-se a expressão a/b formada por primos entre si. Assim, a suposição de que $\sqrt{2}$ é racional, por levar a uma contradição, deve ser abandonada. Aliás, antes de abandonar os pitagóricos, não resisto a mostrar uma demonstração geométrica do teorema de Pitágoras $a^2 + b^2 = c^2$. É ou não é poesia da melhor qualidade ?

9.4.3 Em Resumo

Lógica de Predicados Para passar da lógica de proposições para a lógica de predicados, devemos escrever os predicados. Aqui, a proposição BilharMendes, seria escrita

[4]Introdução à história da matemática, de Howard Eves, 2004, Editora da Unicamp, página 104 a 107.



Bilhar(Mendes), mostrando claramente que Mendes é uma das pessoas que joga bilhar, ou Mendes tem o predicado Bilhar. Na lógica de predicados precisamos de 2 novos símbolos, chamados qualificadores. O Universal é $\forall$ (lê-se qualquer que seja), e o Existencial é $\exists$ (lê-se existe). Introduz-se aqui o conceito de variável, que pode assumir qualquer valor dentro de um domínio especificado.

Eis como ficaria a frase "Quem trabalha duro ou é amigo do chefe, tem promoção"em lógica de predicados:
$\forall$ x Promoção(x) $\Rightarrow$ TrabalhaDuro(x) $\vee$ AmigoChefe(x)
Ou a frase: "Todo funcionário tem um chefe"
$\forall$ x $\exists$ y ÉChefe(y,x).

Há uma distinção aqui: assertivas iniciais do problema, são chamadas axiomas, enquanto conclusões obtidas a partir das regras da lógica e dos axiomas originais, são chamados teoremas. O processo recebe o nome de Modus Ponens, que nada mais é que "Se há uma axioma E1 $\Rightarrow$ E2, e há outro axioma E1, segue-se que E2 é V".

Há também outra regra de inferência chamada Modus Tolens, que diz: Se há um axioma E1 $\Rightarrow$ E2, e há também $\sim$E2, segue-se logicamente que $\sim$E1. (Donde o simbolo $\Rightarrow$ pode ser lido como se e somente se).

Cláusulas Para que os teoremas possam ser provados, todos os axiomas e teoremas precisam estar na fórmula de cláusulas. Para conseguir isso, usam-se as seguintes regras:

1. Eliminar a implicação, lembrando que A $\Rightarrow$ B pode ser escrito como $\sim$A $\vee$ B

2. Reduzir o escopo do $\sim$, lembrando que $\sim\sim$A = A
   $\sim$(A$\wedge$B) = $\sim$A $\vee$ $\sim$B (e vice versa $\sim$(A$\vee$B) = $\sim$A $\wedge$ $\sim$B) (Lei de DeMorgan)
   Exemplos:

   (a) Supondo que P = Aluno aprovado, F = freqüência acima de 75% e N = nota maior que 7. Então P $\Rightarrow$ (F $\wedge$ N). Negando P - o que seria a reprovação, fica: $\sim$P $\Rightarrow$ $\sim$F $\vee$ $\sim$N

   (b) $\sim$ $\forall$ x P(x) = $\sim$P(x) (Não há qualquer um que seja marciano = não há marcianos)

   (c) $\sim$ $\exists$ x P(x) = $\forall$ x $\sim$P(x) (Não existe uma pessoa que seja azul = para qualquer pessoa, ela não será azul)

3. Separar as variáveis que querem dizer coisas diferentes. Como os nomes são só referenciais, isso não altera a cláusula Exemplo: "$\forall$ x P(x) $\vee$ $\forall$ x Q(x) deve ser trocado para $\forall$ x P(x) $\vee$ $\forall$ y Q(y)

 ©88-09, Pedro Kantek

4. Deslocar todos os quantificadores para a esquerda. Isso agora pode ser feito, pois não há mais confusão de nomes, graças ao passo 3.

5. Eliminar o $\exists$, graças a um truque descoberto por Skolem em 1920. Trata-se de introduzir uma função (sobre a qual nada se sabe ainda), mas que permite ir em frente. Por exemplo $\exists$ x Maluco (x), pode ser trocado por Maluco(S1), onde S1 é a função proposta, e que em tempo oportuno dirá V ou F em relação à maluquice. A propósito, o nome da função "S"é homenagem a Skolem.

6. Retirar os quantificadores. Graças às etapas anteriores, isso pode ser feito.

7. Separar as clásusulas unidas por ou em cláusulas unidas por E (graças a DeMorgan).

O resultado final é uma série de cláusulas, cada uma delas unidas por E, que fica implícito.

Processo de resolução A resolução agora é tomar duas cláusulas quaisquer, que resolvidas dão origem a uma nova cláusula. Para tanto, em cada uma das cláusulas juntadas deve haver o mesmo literal ora afirmado ora negado. Esses dois se cancelam, e a cláusula resultante é o que sobrou da anulação.

Se se tentar o processo de resolução por refutação (o mais fácil), deve-se acrescentar a negação do teorema a provar.

Vamos a um exemplo: Seja uma mesa que tem 2 tijolos, um sobre o outro. O de baixo é A, e o de cima é B. Claramente pode-se afirmar que Sobre(B,A) é V, e que Sobre(A,Mesa) também é V. Queremos provar que B está encima da mesa. Ou, Acima(B,mesa).

Axiomas: $\forall$ x $\forall$ y [Sobre(x,y) $\Rightarrow$ Acima(x,y)]
$\forall$ x $\forall$ y $\forall$ z [Acima(x,y) $\wedge$ Acima(y,z) $\Rightarrow$ Acima(x,z)]

Percorrendo os procedimentos acima, os axiomas ficam $\sim$Sobre (u, v) $\vee$ Acima (u, v)
$\sim$Acima(x, y) $\vee$ $\sim$Acima(y, z) $\vee$ Acima(x, z)

Queremos provar Acima(B, mesa), logo introduzimos $\sim$Acima (B, mesa). Fica o conjunto
(1) $\sim$Sobre (u, v) $\vee$ Acima (u, v)
(2) $\sim$Acima (x, y) $\vee$ $\sim$Acima(y, z) $\vee$ Acima (x, z)
(3) Sobre (B, A)
(4) Sobre (A, mesa)
(5) $\sim$Acima (B, mesa)

Tomando (2) e (5) fazendo x = B e z = mesa, juntando e anulando, fica
(6) $\sim$Acima(B, y) $\vee$ $\sim$Acima(y, mesa)

Tomando (1) e (6) fazendo u = y e v = mesa, fica
(7) $\sim$Sobre (y, mesa) $\vee$ $\sim$Acima (B, y)

Tomando (1) com (7) fazendo u = B e v = y, fica
(8) $\sim$Sobre (B, y) $\vee$ $\sim$Sobre (y, mesa)

Tomando (3) e (8) fazendo y = A
(9) $\sim$Sobre (A, mesa)

Tomando (4) e (9), ambas se anulam e o resultado é NIHIL. Logo, a assertiva negada estava correta.

Outro Exemplo

Seja a seguinte situação

> João tem um cachorro. Todos os proprietários de cachorros amam os animais. Ninguém que ame os animais os mata. João ou a Curiosidade matou um gato, que se chamava Tuna. A Curiosidade matou o gato ?

Escrevendo as orações originais e usando algum senso comum (um gato é um animal...)

1. Dono (x,João) $\wedge$ Cachorro(x)
2. Dono (x,y) $\wedge$ Cachorro(x) $\wedge$ Animal(z) $\Rightarrow$ Ama (y,z)
3. Ama (x,z) $\Rightarrow \sim$ Mata(x,z)
4. Mata(João,Tuna) $\vee$ Mata(Curiosidade,Tuna)
5. Cachorro(x) $\Rightarrow$ Animal(x)
6. Animal(Tuna)

Note que a frase 1 deve ser desdobrada em 1.1 e 1.2 e Reescrevendo a frase 2. $\sim$[Dono (x,y) $\wedge$ Cachorro(x) $\wedge$ Animal(z)] $\vee$ Ama (y,z)
fica:

1.1 Dono (x,João)
1.2 Cachorro(x)
2. $\sim$ [Dono (x,y) $\vee \sim$ Cachorro(x) $\vee \sim$ Animal(z)] $\vee$ Ama (y,z)
3. $\sim$ Ama (x,z) $\vee \sim$ Mata(x,z)
4. Mata(João,Tuna) $\vee$ Mata(Curiosidade,Tuna)
5. $\sim$ Cachorro(x) $\vee$ Animal(x)

6. A Provar: $\sim$ Mata(Curiosidade,Tuna)

De 6 e 4, fica: $\sim$ Mata(Curiosidade,Tuna) + Mata(João,Tuna) $\vee$ Mata(Curiosidade,Tuna) e simplificando,
7. Mata(João,Tuna)

De 7 e 3, substituindo x/João e y/Tuna, fica: Mata(João,Tuna) + $\sim$ Ama (João,Tuna) $\vee \sim$ Mata(João,Tuna) e simplificando
8. $\sim$ Ama (João,Tuna)

De 8 e 2, substituindo y/João, z/Tuna, fica:
$\sim$ Ama (João,Tuna) + $\sim$ Dono (x,João) $\vee \sim$ Cachorro(x) $\vee \sim$ Animal(Tuna)] $\vee$ Ama (João,Tuna) e simplificando
9. $\sim$ Dono (x,João) $\vee \sim$ Cachorro(x) $\vee \sim$ Animal(Tuna)

De 9 e 1.1, fica $\sim$ Dono (x,João) $\vee \sim$ Cachorro(x) $\vee \sim$ Animal(Tuna) + Dono (x,João) e simplificando,
10. $\sim$ Cachorro(x) $\vee \sim$ Animal(Tuna)

De 10 e 1.2, fica
$\sim$ Cachorro(x) $\vee \sim$ Animal(Tuna) + Cachorro(x), e simplificando
11. $\sim$ Animal(Tuna)

Finalmente, de 11 e 6, temos: 12. $\sim$ Animal(Tuna) + Animal(Tuna), que se anulam e o resultado é NIL.

A questão agora é mostrar que Mata(Curiosidade,Tuna) é verdadeira. Para resolver suponhamos a negação da meta e aplicando a regra de inferência sete vezes, chega-se à uma contradição, o que significa que a meta é verdadeira.

Em português, a resolução poderia ser expressa como

> Suponha-se que Curiosidade não matou a Tuna. Sabemos que o assassino era João ou a Curiosidade, então foi o João. Mas, se João é dono de D e D é um cachorro, então João ama os animais. Tuna é um gato e os gatos são

 ©88-09, Pedro Kantek

animais, logo Tuna é um animal. Os amantes dos animais não os matam, pelo que João não matou Tuna. Aqui a contradição, pois João matou Tuna e João não matou Tuna. Assim, prova-se que a Curiosidade matou Tuna.

9.4.4 Exemplo

Seja o seguinte enunciado

> Marcos era um homem. Marcos era um pompeu. Os pompeus eram romanos. Cesar era um soberano. Todos os romanos eram leais a Cesar ou odiavam Cesar. Cada pessoa é leal a alguém. Pessoas tentam assassinar soberanos a quem não sejam leais. Marcos tentou assassinar a Cesar. Finalmente, todo homem é uma pessoa. Pergunta-se: Marcos odiava Cesar ?

Escrevendo o texto na forma de cláusulas: Aqui estão as declarações traduzidas para a linguagem da lógica
1. Homem(M)
2. Pompeu(M)
3. $\forall x$ Pompeu(x) $\Rightarrow$ Romano(x)
4. Soberano(C)
5. $\forall x$ Romano(x) $\Rightarrow$ Leal(x,C) $\vee$ Odeia(x,C)
6. $\forall x \exists\, y$ Leal(x, y)
7. $\forall x \exists\, y$ Pessoa(x) $\wedge$ Soberano(y) $\wedge$ Tentamatar(x, y) $\Rightarrow \sim$ Leal(x, y)
8. Tentamatar(M,C)
9. $\forall x$ Homem(x) $\Rightarrow$ Pessoa(x)

Excluindo $\Rightarrow$: Considerando que $A \Rightarrow B$ pode ser trocado por $\sim A \vee B$ e reescrevendo as linhas 3,5,7 e 9 fica
3. $\forall x \sim$ Pompeu(x) $\vee$ Romano(x)
5. $\forall x \sim$ Romano(x) $\vee$ Leal(x,C) $\vee$ Odeia(x,C)
7. $\forall x \exists\, y \sim$[Pessoa($x$) $\wedge$ Soberano(y) $\wedge$ Tentamatar(x, y)] $\vee \sim$ Leal(x, y)
9. $\forall x \sim$ Homem(x) $\vee$ Pessoa(x)

Padronizando nomes: E passando os quantificadores para a esquerda, fica:
1. Homem(M)
2. Pompeu(M)
3. $\forall x_1 \sim$ Pompeu(x_1) $\vee$ Romano(x_1)
4. Soberano(C)
5. $\forall x_2 \sim$ Romano(x_2) $\vee$ Leal(x_2,C) $\vee$ Odeia(x_2,C)
6. $\forall x_3 \exists\, y$ Leal(x_3, y)
7. $\forall x_4 \exists\, y \sim$[Pessoa($x_4$) $\wedge$ Soberano(y) $\wedge$Tentamatar(x_4, y)] $\vee \sim$ Leal(x_4, y)
8. Tentamatar(M,C)
9. $\forall x_5 \sim$ Homem(x_5) $\vee$ Pessoa(x_5)

Retirando os quantificadores: e aplicando a função de Skolem.
1. Homem(M)
2. Pompeu(M)
3. $\sim$ Pompeu(x_1) $\vee$ Romano(x_1)
4. Soberano(C)
5. $\sim$ Romano(x_2) $\vee$ Leal(x_2,C) $\vee$ Odeia(x_2,C)
6. Leal($x_3, S_1(y)$)
7. $\sim$[Pessoa(x_4) $\wedge$ Soberano(y) $\wedge$ Tentamatar(x_4, y)] $\vee \sim$ Leal(x_4, y)
8. Tentamatar(M,C)
9. $\sim$ Homem(x_5) $\vee$ Pessoa(x_5)

Aplicando $\sim$ à frase 7
7. $\sim$ Pessoa(x_4) $\vee \sim$ Soberano(y) $\vee \sim$ Tentamatar(x_4, y) $\vee \sim$ Leal(x_4, y)

Para responder: À pergunta Marco odeia Cesar ?, vamos introduzir a frase negada
10. $\sim$ Odeia(M,C)

Juntando 5 e 10, e substituindo x_2 por Marcos, fica:
$\sim$ Romano(M) $\vee$ Leal(M,C) $\vee$ Odeia(M,C) + $\sim$ Odeia(M,C), simplificando, fica
11. $\sim$ Romano(M) $\vee$ Leal(M,C)

Juntando a 11 com a 3 e substituindo x_1 por Marcos, fica
$\sim$ Romano(M) $\vee$ Leal(M,C) + $\sim$ Pompeu(M) $\vee$ Romano(M), simplificando, fica
12. Leal(M,C) $\vee \sim$ Pompeu(M)

Juntando a 12 com a 2, fica
Leal(M,C) $\vee \sim$ Pompeu(M) + Pompeu(M) e simplificando, fica
13. Leal(M,C)

Juntando a 13 com a 7, e substituindo x_4 por M e y por C, fica:
Leal(M,C) + $\sim$ Pessoa(M) $\vee \sim$ Soberano(C) $\vee \sim$ Tentamatar(M,C) $\vee \sim$ Leal(M,C)
e simplificando:
14. $\sim$ Pessoa(M) $\vee \sim$ Soberano(C) $\vee \sim$ Tentamatar(M,C)

Juntando a 14 com a 4, fica
$\sim$ Pessoa(M) $\vee \sim$ Soberano(C) $\vee \sim$ Tentamatar(M,C) + Soberano(C) e simplificando
15. $\sim$ Pessoa(M) $\vee \sim$ Tentamatar(M,C)

Juntando a 15 com a 8, fica $\sim$ Pessoa(M) $\vee \sim$ Tentamatar(M,C) + Tentamatar(M,C) e simplificando, fica
16. $\sim$ Pessoa(M)

Juntando a 16 com a 9 e substituindo x_5 por Marcos, fica
$\sim$ Pessoa(M) + $\sim$ Homem(M) $\vee$ Pessoa(M) e simplificando
17. $\sim$ Homem(M)

Finalmente, juntando a 17 com a 1, fica
18. $\sim$ Homem(M) + Homem(M), e o resultado é NIL.
Logo demonstra-se que Marcos odiava Cesar.

9.4.5 Exercícios

Transforme a situação em uma base de conhecimento normalizada

1. Cavalos, vacas e porcos são mamíferos
2. O descendente de um cavalo é um cavalo
3. Barbaazul é um cavalo
4. descendente e progenitor são relações inversas
5. Todo mamífero tem um progenitor

 ©88-09, Pedro Kantek

9.4.6 Resolução na LPO

Dois literais serão contraditórios se um deles puder ser unificado com a negação do outro. Assim por exemplo Gato(x) e ∼Gato(Rafaela) são contraditórios, pois Gato(x) e Gato(Rafaela) podem ser unificados. Isso corresponde à intuição que diz que não pode ser verdadeiro para todos os x que Gato(x) se houver conhecimento de algum x, por exemplo, a Rafaela, para a qual Gato(x) é falso.

Para resolver expressões de lógica de predicados usar-se-á o algoritmo da Unificação para localizar pares de literais que cancelem uns aos outros.

Também será usado o unificador produzido pelo algoritmo da unificação para gerar a cláusula resolvente. Por exemplo, supondo querer resolver duas cláusulas

1. Homem(Marco)
2. ∼Homem(x_1) ∨ Mortal(x_1)

O literal Homem(Marco) pode ser unificado com o literal Homem(x_1) com a substituição Marco/x_1 dizendo-nos que para x_1 = Marco, Homem(Marco) e ∼Homem(Marco) se cancelam gerando o resolvente Mortal(x_1)

9.4.7 Exemplo

```
O problema do Asterix
c  : Asterix está com o seu chapeu sobre a cabeça
co : Asterix come
d  : Asterix dorme
g  : Asterix espera o glu-glu
a  : Asterix está aqui

(~c) <=> (co OU d) (0)     Asterix só tira o chapeu para comer e dormir
g => ~co                   não come pois esperava os glu-glus
d => a, como ~a => ~d      Se Asterix dormisse, estaria aqui. Como não
                           está... (Modus Tollens)

Dificuldades: 1. formalizar o discurso usual
2. Descobrir as premissas implícitas
Resolução

g com g=>~co, resulta ~co  (1)
d=>a  e ~a, resulta ~d (2)
(1) e (2) resultam ~co E ~d  (3)
(3), por Morgan resulta ~(co OU d) (4)
(4) e (0) resulta c (5)
Como ~c e (5), resulta NIHIL
```

Desafio

- A história s seguir foi extraída do livro de Niklaus Wirth "Algorithms + Data Structures = programs".

  > Eu me casei com uma viúva (vamos chamá-la de V) que tem uma filha adulta (cujo nome é A). Meu pai (P), que nos visitava freqüentemente, se apaixonou pela minha enteada e a desposou. Assim, meu pai se tornou meu genro e minha enteada se tornou minha mãe. Alguns meses mais tarde, a minha esposa deu a luz a um filho (F_1) que se tornou o cunhado

de meu pai, bem como meu tio. A esposa de meu pai, minha enteada, também teve um filho (F_2).

Usando cálculo de predicados, crie um conjunto de expressões que representem a situação relatada na história acima. Acrescente expressões que definam as relações familiares básicas como a definição de sogro e empregue o *modus ponens* sobre este sistema para provar a conclusão "eu sou meu próprio avô".

9.5 Apêndice

9.5.1 Unificação em LISP

O algoritmo de unificação atribui valores para as variáveis em uma fórmula.

```
1: Algoritmo UNIFY (L1, L2) {(como definido por Robinson, 65)}
2: se L1 ou L2 é átomo então
3:    se se L1 = L2 então
4:       retorna NIL
5:    senão
6:       se L1 é variável então
7:          se se L1 ocorre em L2 então {teste ***}
8:             retorna (falha)
9:          senão
10:            Retorna (L1.L2) {ou (L1/L2)}
11:         fimse
12:      fimse
13:      se L2 é variável então
14:         se L2 ocorre em L1 então
15:            Retorna (falha)
16:         senão
17:            Retorna (L2.L1) {ou (L2/L1)}
18:         fimse
19:      fimse
20:   fimse
21: senão
22:   se comprimento(L1) ≠ comprimento (L2) então
23:      retorne(falha)
24:   fimse
25:   subst ← nil
26:   Seja L1 = (L11, L12, ..., L1n) e L2 = (L21, L22, ..., L2n)
27:   para i = 1 ... n faça
28:      s ← UNIFY (L1i, L2i)
29:      se s = falha então
30:         retorne (falha)
31:      fimse
32:      Aplique s → L1
33:      Aplique s → L2
34:      subst ← append (s subst)
35:   fimpara
36:   retorne (subst)
37: fimse
```

O teste *** é feito para evitar a substuição (x.x) ou que é pior (x.f(x))
Exemplo do pior caso do algoritmo de Robinson

 ©88-09, Pedro Kantek

```
Seja unificar f(u, h(w,w), w, j(y,y)) e f(g(v,v), v, i(x,x), x))

Unificador é
x  j(y,y)
w  i(x,x)
v  (h(w,w)
u  g(v,v); e fica

e fica
x  j(y,y)
w  i(j(y,y),j(y,y))
v  h(i(j(y,y),j(y,y),i(j(y,y),j(y,y)))
u  g(h(i(j(y,y),j(y,y),i(j(y,y),j(y,y))),h(i(j(y,y),j(y,y),i(j(y,y),j(y,y))))
```

O problema aqui é a instanciação de variáveis no unificador durante a construção da própria substituição.

Moral da história: a substituição vai devolver uma lista de listas, onde cada elemento é um par. Do lado esquerdo tenho uma variável e do lado direito tenho um valor. Aqui entra em ação um macete do LISP. Quando sei que a minha lista tem só dois elementos, em vez de LIST (que pode ter 0, 1, 2, ... n elementos) uso apenas o cons que só pode ter 2 elementos (separados por ponto).

```
1: Algoritmo UNIFY em LISP
2: (defun unify (L1 L2)
3: (cond ((or (atom L1)(atom L2))(cond ((eql L1 L2) nil)
4: ((and (var-p L1) (free-p L2 L1)) (list (cons L1 L2)))
5: ((and (var-p L2) (free-p L1 L2)) (list (cons L2 L1)))
6: (T 'fail )))
7: ((= (length L1) (length L2)) (let ((s (unify (car L1) (car L2)))
8: (cond ((eql s 'fail) 'fail)
9: (T
10: (let ((subst (unify (apply-subst s (cdr L1))
11: (apply-subst s (cdr L2)))
12: (cond ((eql subst 'fail) fail)
13: (T (append s subst ))))))))
14: (T 'fail)))
```

As hipóteses de L1 e L2 são: um dos dois é átomo (linhas 1 a 4), ambos tem comprimento igual (5 a 11) ou falha (linha 12).

Se um dos dois é átomo e eles são iguais, retorna-se nil (1) e o programa acaba.

Se um dos dois é átomo, eles são diferentes e L1 é variável e L1 não está em L2, retorna-se a lista formada pelo cons de L1 e L2. (2)

Se um dos dois é átomo, eles são diferentes e L2 é variável e L2 não está em L1, retorna-se a lista formada pelo cons de L2 e L1. (3)

Se um dos dois é atomo, e nada disso ocorreu, retorna-se fail (4)

Se L1 e L2 são listas, tem o mesmo tamanho, cria-se o par s que é a unificação entre o car de L1 e o car de L2, ou é fail. (note a recursividade) (5)

Se s é fail, retorna-se fail (6)

Se s não é fail, cria-se a variável subst, que é a unificação entre (a aplicação da substituição de s no cdr de L1) e (a aplicação da substituição de s no cdr de L2) (linhas 7, 8 e 9)

Se subst é fail, retorna-se fail (10)

Se subst não é fail, appenda-se subst a s (11)

Se nem L1 nem L2 são átomos, nem tem o mesmo tamanho, retorna-se fail (12)

Obs: Vide capítulo 2 do Luger.

 ©88-09, Pedro Kantek

Capítulo 10

Como representar conhecimento

A primeira aposta foi a lógica, que como vimos nas aulas anteriores, tem lá o seu valor. Os problemas começam a surgir quando se mapeia uma linguagem natural, como o português diretamente em especificações lógicas. É razoável mapear os operadores $\vee$ a **ou**, ou $\Rightarrow$ a **se... então**. Entretanto na lógica está-se preocupado apenas nos valores verdade e ignora-se que muitas vezes a construção **se... então** do português sugere um relacionamento específico, freqüentemente mais correlacional do que causal. [1] Seja por exemplo a frase *se uma ave é um urubú, então ela é preta.* Escrevendo isso em cálculo de predicados, tem-se

$$\forall x \ (\text{urubu}(x) \Rightarrow \text{preto}(x).$$

Usando as ferramentas da lógica, a expressão acima pode ser reescrita na sua expressão equivalente

$$\forall x \ (\sim \text{preto}(x) \Rightarrow \sim \text{urubu}(x)).$$

O fato das duas expressões serem equivalentes implica em que a segunda é verdadeira se e somente se a primeira o for. Entretanto, e claramente, a equivalencia do valor verdade é inapropriada neste caso. Siga o raciocínio:

1. A parede à sua frente não é preta
2. A parede à sua frente não é um urubú
3. Isto é uma evidência para a verdade da segunda expressão
4. Como as duas expressões são equivalentes, a evidência também o é para a primeira expressão.
5. Finalmente, conclui-se que a brancura da parede é uma evidência de que urubús são pretos (o que é sem sentido e bobo...)

A razão para esta incongruência é que a implicação lógica expressa apenas uma relação entre valores verdade de seus operandos, enquanto a sentença em português implica uma correlação positiva entre a pertinência a uma classe e a posse de propriedades dessa

[1] Em um de seus livros, Carl Sagan dá um exemplo claro do tipo de conclusão bizarra a que se pode chegar quando se confunde correlação e causa: "um levantamento mostra que é maior o número de homossexuais entre os que têm curso superior do que entre os que não o possuem; portanto, a educação torna as pessoas homossexuais."

classe. A disposição genética do urubú faz com que ele seja preto. Este fato é perdido na segunda versão da expressão. Embora o fato da parede não ser preta seja consistente com a verdade em ambas as sentenças, ele é irrelevante para a natureza causal da cor dos pássaros.

As teorias *associacionistas* definem o significado de um objeto em termos de uma rede de associações com outros objetos[2]. Para os que defendem estas teorias, quando os seres humanos percebem um objeto e raciocinam sobre ele, esta percepção é mapeada primeiro para um conceito. Este conceito é parte do nosso conhecimento de mundo e está conectado através de relacionamentos apropriados a outros conceitos. Estes relacionamentos formam uma compreensão das propriedades e comportamento de objetos.

Há evidências psicológicas de que, além de sua habilidade para associar conceitos, os humanos também organizam hierarquicamente o seu conhecimento, de forma que a informação seja mantida nos níveis apropriados mais altos da taxonomia. Quillian e Collins, a partir de 1966, modelaram o armazenamento e o gerenciamento da informação pelo homem usando uma rede semântica. A estrutura dessa hierarquia foi derivada a partir de testes de laboratório de pessoas humanas. Perguntava-se à elas sobre diferentes propriedades dos pássaros, tas como *um canário é um pássaro ?* ou *um canário pode cantar?* ou *um canário pode voar?*.

Por óbvias que sejam as respostas a essas questões, o pesquisador estudou o tempo médio em que as pessoas respondiam a cada pergunta e verificou que a demora era maior para responder *um canário pode voar?* do que a pergunta *um canário pode cantar?*. Os dois pesquisadores explicaram as diferenças de tempo argumentando que as pessoas armazenam informações no seu nível mais abstrato. Em vez de recordar que canários voam, que pardais voam e que andorinhas voam, tudo armazenado com o pássaro específico, o homem recorda que canários são pássaros e que pássaros tem (normalmente) a propriedade de voar. Propriedades mais gerais como alimentação, respiração e movimentação são armazenadas no nivel do animal e assim tentar recordar se um canário pode respirar deve levar mais tempo do que recordar se um canário pode voar. Isto ocorre porque o recordante deve avançar mais acima na hierarquia das estruturas de memória para encontrar a resposta. A recordação mais rápida se dava nos traços específicos do pássaro ou seja, de que ele sabe cantar e é amarelo. O tratamento de exceções também ocorre no nível mais específico. Quando se perguntava se um avestruz pode voar a resposta era mais rápida do que quando se perguntava se uma avestruz pode respirar. Assim, a hierarquia avestruz → pássaro → animal parece não ser atravessada para se chegar à informação sobre a exceção. Ela está diretamente na avestruz.

Sistemas baseados em herança (como as redes semânticas) permitem armazenar a informação no nível mais alto de abstração. Isto reduz o tamanho da base e ajuda a prevenir inconsistências na sua atualização.

Assim, além do trabalho com a lógica, uma linha alternativa se desenvolveu dos esforços de psicólogos e linguistas para caracterizar a natureza da compreensão humana. Aqui não existe a preocupação de estabelecer uma ciência do raciocínio correto mas em descrever o modo como seres humanos adquirem e usam conhecimento do mundo. Esta pesquisa tem se mostrado importante em áreas da IA como compreensão de linguagem natural e raciocínio do senso comum.

10.0.2 Redes Semânticas

Uma rede semântica pode ter inúmeras definições, em geral, uma para cada autor importante na área. A definição que será usada aqui é a de um grafo onde os nós correspondem

[2]Isto já foi notado por Aristóteles que dizia ser uma definição uma soma entre *genus* e *diferentia*. Por exemplo: O Brasil é um país grande. O *genus* é "país", enquanto que a *diferentia* é "grande".

a conceitos ou fatos e os arcos a relações ou associações entre conceitos ou fatos. Tanto arcos quanto nodos, devidamente rotulados.

Talvez o trabalho mais antigo a influenciar as RS foi o sistema de grafos existenciais de Charles Peirce (1839, 1914),[3] ainda no século 19. A teoria de Peirce tinha todo o poder expressivo do cálculo de predicados de primeira ordem, com uma base axiomática e regras de inferência formais. Em 1896, Charles Peirce sugeriu o que chamou de "grafos existenciais "aos quais chamou de "a lógica do futuro". Isto começou um longo debate entre os que advogam a lógica e os adeptos das redes semânticas. Hoje é aceito que ambos são equivalentes. O importante é que a ferramente deve ajudar a compreender a semântica. A síntaxe é menos importante. Se usamos strings (lógica) ou desenhos (snets+frames) é indiferente, mas não o é para a compreensão.

Muito da pesquisa em representações em redes tem sido na área de compreensão de linguagem. Em geral, esta aplicação, mesmo em domínios restritos, requer muito mais conhecimento do que, por exemplo, os sistemas especialistas. Aqui é necessário compreensão do senso comum, dos modos como os objetos físicos se comportam, as iterações que ocorrem entre seres humanos e os modos como as instituições humanas estão organizadas. Para pretender entender a linguagem um programa precisa compreender intenções, crenças, raciocínios, inclusive hipotéticos, planos e objetivos. Como fazer isso requer grandes quantidades de conhecimento, este problema (a compreensão de linguagem natural) tem sido uma força na busca de esquemas de representação do conhecimento.

As primeiras implementações de redes semânticas em computador foram desenvolvidas no início dos anos 60 para uso em tradutores automáticos. Em 1961, Masterman definiu um conjunto de 100 tipos de conceitos primitivos e os usou para definir um dicionário de 15.000 conceitos.

Planos de Quillian

Um programa importante em RS foi escrito por Quillian em finais dos 60. Ele define as palavras em inglês de maneira parecida à de um dicionário. Os componentes da definição (outras palavras) são definidos da mesma forma. Em vez de definir as palavras através de axiomas formais (abordagem da lógica), cada definição simplesmente conduz a outras de maneira meio desestruturada e certamente circular. Ao procurar entender uma palavra, deve-se percorrer a rede até estar-se satisfeito com os termos encontrados. A base de conhecimentos estava organizada em planos e cada plano era um grafo que definia uma única palavra. A figura 10.1 ilustra 3 planos que descrevem acepções diferentes da palavra *plant* (em inglês): são elas, o substantivo planta, uma fábrica e o verbo plantar. A definição original de cada grafo é: um organismo vivo (acepção 1), um lugar onde pessoas trabalham (a 2) e o ato de colocar uma semente no solo (3). O programa usa esta base de conhecimentos para encontrar relacionamentos entre pares de palavras em inglês. Dadas duas palavras, ele busca em amplitude nos grafos que se originam de cada palavra, buscando um conceito comum ou nó de intersecção. Os caminhos para os nós representam um relacionamento entre os conceitos da palavra. A figura 10.2 ainda do artigo original de Quillian representa os caminhos de intersecção entre chorar e confortar. Usando esse caminho, o programa era capaz de concluir: (os números entre parênteses indicam qual a acepção usada).

> chorar (2) é, entre outras coisas, fazer um som triste. Confortar (3) pode ser fazer (2) um pouco menos triste. (Quillian, 1967).

[3] Este americano surge como o iniciador da semiótica, mas ele foi também matemático, astrônomo, químico, geodésico, agrimensor, cartógrafo, metrologista, espectroscopista, engenheiro, inventor, psicólogo, filologista, lexicógrafo, historiador da ciência, economista, estudante de medicina, revisor de livros, ator, escritor de contos, fenomenologista, lógico, retórico e metafísico, bacana o cara, não?

 ©88-09, Pedro Kantek

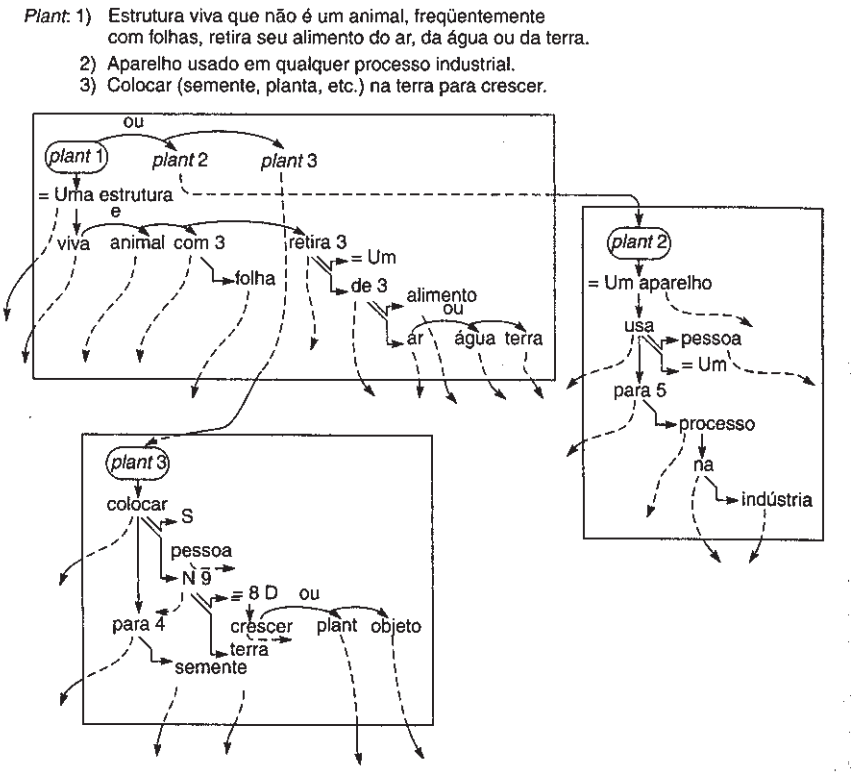


Figura 10.1: Três definições da palavra *plant*

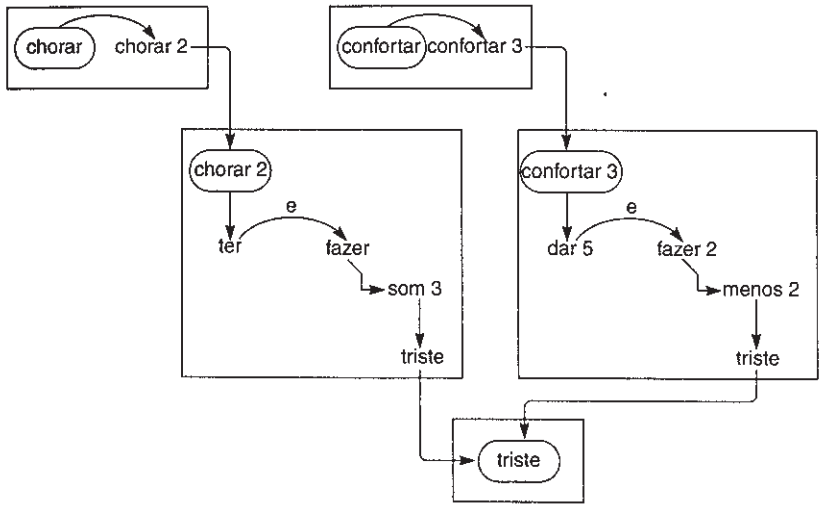


Figura 10.2: Caminho de intersecção entre chorar e confortar

Quillian sugeria que esta abordagem para semântica poderia dotar um sistema de compreensão da linguagem natural com capacidades de:

1. Determinar o significado de um trecho de texto por meio da construção de uma

coleção de nós de intersecção

2. Escolher entre múltiplos significados de palavras, buscando o significado como sendo o menor caminho de intersecção para outras palavras da sentença

3. Responder a uma gama flexível de consultas com base em associações entre conceitos das palavras das consultas e os conceitos do sistema

Eis um exemplo do segundo item acima: Na frase *Tom foi para casa regar a sua nova planta*, a acepção correta de planta poderia ser obtida através da intersecção dos conceitos das palavras regar e planta.

Este trabalho inicial foi importante, mas ele deixava em aberto a tipificação das arestas. Padecia portanto de uma grande generalidade de aplicação, o que acabou limitando sua utilidade. Se de um lado qualquer tipo de relacionamento poderia ser construído, de outro ele transfere para o programador o (pesado) fardo de construir conjuntos apropriados de fatos e regras.

Ou seja, para obter engenhos mais consistentes, é necessário que os relacionamentos sejam formais e passem a fazer parte do formalismo.

Frames de Casos de Simmons

Simmons (1973) criou relacionamentos padrão focando os verbos do inglês. Aqui, os elos definem os papéis assumidos por nomes e frases nominais na ação da sentença. Entre os relacionamentos se incluem agente, objeto, instrumento, localização e tempo. Uma sentença é representada como um nó verbal com vários elos de caso ligando nós que representam outros participantes da ação. Esta estrutura é chamada *frame de caso*. Ao achar um verbo o programa recupera o frame de caso para este verbo na sua base de conhecimento. Ele então liga os valores de agente, objeto, aos nós apropriados no frame de caso. Assim, a frase "Sara consertou a cadeira com cola", poderia ser representado pelo frame da figura 10.3

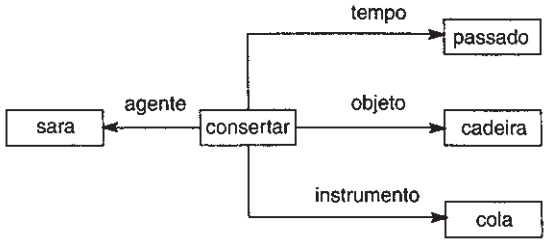


Figura 10.3: Frame de caso para a sentença "Sara consertou a cadeira com cola"

Usando esta estratégia captura-se muito da estrutura mais profunda da linguagem natural, tal como a relação entre um verbo e seu objeto. O conhecimento da estrutura da língua é parte do prórpio formalismo da rede. Quando a sentença individual é analisada, esses relacionamentos embutidos indicam que Sara é a pessoa que executa o conserto e que é usada cola para montar a cadeira. Note-se que estes relacionamentos lingüísticos estão armazenados de uma forma que é independente da sentença real ou mesmo da linguagem na qual a sentença é expressa.

Dependência conceitual de Schank

Esta teoria, de 1974, fornece um conjunto de quatro conceitualizações primitivas das quais o mundo dos significados é construído. Esses conceitos são iguais e independentes.

 ©88-09, Pedro Kantek

São eles:

ATOs Ações

PFs Objetos (produtores de figuras)

AAs Modificadores de ações (auxiliares de ações)

AFs Modificadores de objetos (auxiliares de figuras)

Assume-se que todas as ações se reduzem a um ou mais **ATOs** primitivos. Existe uma lista de primitivas tomadas como componentes básicos de ações com verbos mais específicos sendo formados através de sua combinação e modificação

ATRANS transferir um relacionamento (ex: DAR)

FTRANS transferir a localização física de um objeto (ex: IR)

PROPUL aplicar força física a um objeto (ex: EMPURRAR)

MOVER mover parte do corpo do sujeito (ex: CHUTAR)

PEGAR pegar um objeto – por um agente (ex: AGARRAR)

INGERIR ingerir um objeto – por um animal (ex: COMER)

EXPELIR expelir do corpo de um animal (ex: GRITAR)

MTRANS transferir informação mental (ex: CONTAR)

MCONST construir mentalmente informação nova (ex: DECIDIR)

CONC conceitualizar, refletir sobre uma idéia (ex: PENSAR)

FALAR produzir som (ex: DIZER)

ATENTAR focar órgão sensorial (ex: ESCUTAR)

Essas primitivas são usadas para definir os relacionamentos de dependência conceitual que descrevem estruturas de significado como relação de caso ou associação de objetos e valores. Relacionamentos de dependência conceitual são *regras de síntaxe conceituais* e constituem uma gramática de relacionamentos semânticos significativos. Esses relacionamentos podem ser usados para construir uma representação interna de uma sentença em linguagem natural. Uma lista de dependências conceituais aparece na figura 10.4 Veja mais alguns exemplos em 10.5

10.0.3 Quadros

Um quadro é uma estrutura de dados explicitamente organizada que serve para capturar as conexões implícitas da informação num domínio de problema. Essa representação possibilita a organização do conhecimento em unidades mais complexas que refletem a organização de objetos no domínio. Em um artigo de 1975, Marvin Minsky descreve um quadro (frame)

> Eis aqui a essência da teoria dos frames: Quando alguém encontra uma nova situação (ou modifica substancialmente o seu entendimento sobre um problema) recupera da memória uma estrutura chamada frame. Essa estrutura é um arcabouço memorizado que deve ser adaptado para se adequar à realidade alternando detalhes, conforme a necessidade (Minsky, 1975)

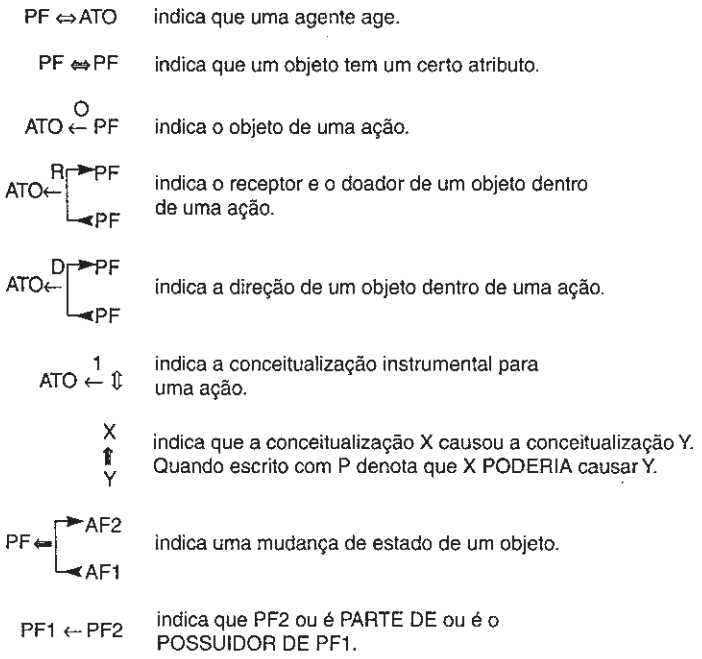


Figura 10.4: Dependências conceituais de Schank

Parece que estruturas similares a quadro nos permitem organizar nosso conhecimento sobre o mundo. Utilizamos informação estruturada por experiências passadas para nos ajustarmos a cada nova situação. Então, moldamos ou revisamos os detalhes dessas experiências passadas para representar as diferenças individuais para a nova situação.

Qualquer um que tenha se hospedado em um ou dois hotéis não encontra problemas ao lidar com hoteis totalmente diferentes. Os detalhes deste caso (o que estamos estudando) são preenchidos sobre aquele quadro. Não precisamos reconstruir o nosso entendimento a cada novo quarto de hotel que ocupamos. Veja por exemplo, na figura 10.6 um frame representando um quarto de hotel.

Cada frame pode ser visto como uma estrutura de dados similar em vários aspectos ao registro tradicional (ou até ao objeto tradicional), que contém informação relevante acerca de entidades estereotipadas. Os componentes do frame contém informações como

1. Informação sobre a identificação do frame
2. Relação deste frame com outros frames
3. Informação procedimental sobre o uso da estrutura descrita
4. defaults para atributos do frame
5. etc

Os frames tornam mais fácil tratar hierarquicamente o conhecimento. Nas RS cada conceito é representado por nós e elos no mesmo nivel de especificação. É comum, entretanto, podemos querer considerar um objeto como entidade única para certos propósitos e considerar detalhes de sua estrutura interna para outros propósitos.

 ©88-09, Pedro Kantek

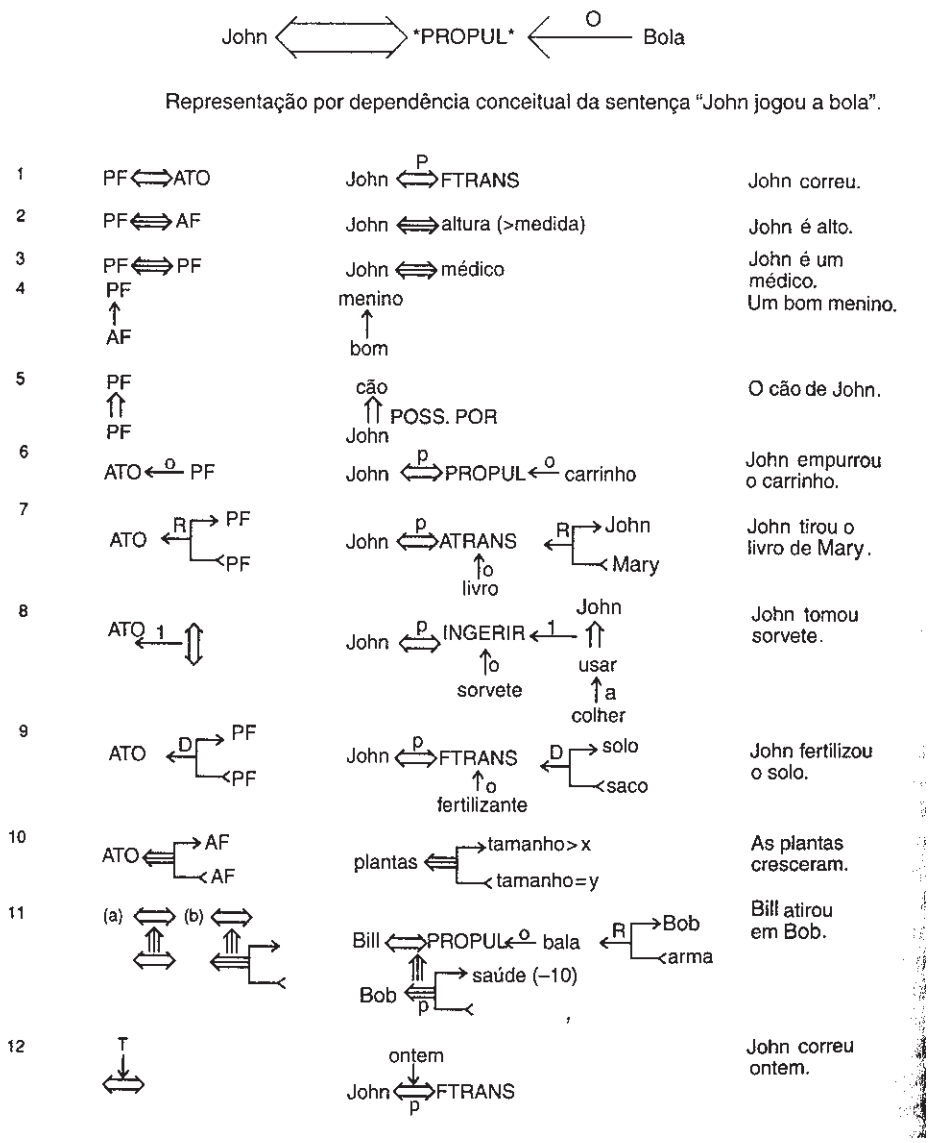


Figura 10.5: Exemplos de dependências conceituais de Schank

Por exemplo, nosso conceito de carro pressupõe uma máquina que nos transporta. É uma entidade única. Entretanto, quando ele pifa, é necessário baixar a abstração e adentrar no funcionamento, componentes, diagnósticos, partes e sistemas do carro.

Sistemas de quadros suportam herança de classe. Os componentes e defaults de um frame são herdados através da hierarquia classe/subclasse e classe/membro. Quando um caso de um frame é criado, o sistema tentará preencher seus componentes perguntando ao usuário, aceitando o valor default do frame de classe ou executando algum procedimento previamente determinado para obter o valor para o caso.

Frames superam o poder das redes semânticas por permitir que objetos complexos sejam representados como um único frame em vez de uma grande estrutura de rede.

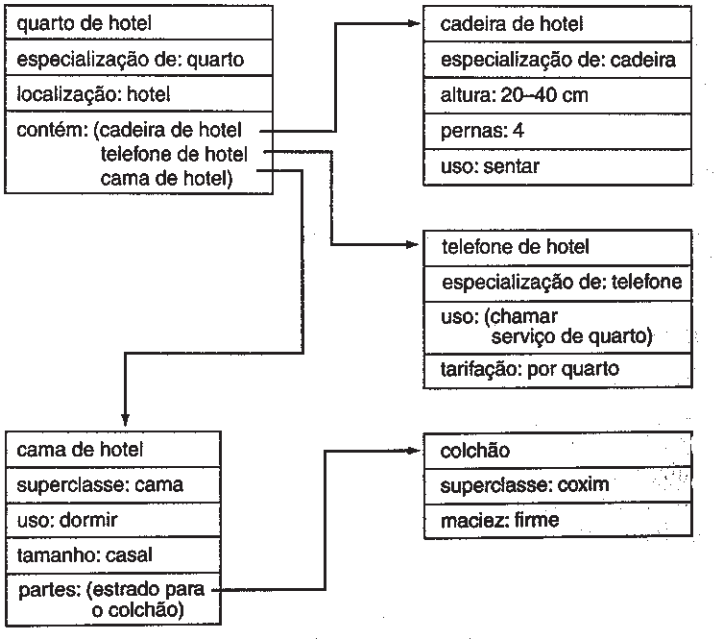


Figura 10.6: Parte de um frame descrevendo um quarto de hotel

Isso fornece também um meio natural de representar entidades estereotipadas, classes, herança, valores default. Embora os frames, como as representações por lógica e por rede sejam uma ferramenta poderosa, muitos dos problemas de adquirir e organizar uma base de conhecimento complicada devem ainda ser resolvidos pela habilidade e intuição do programador.

10.0.4 Grafos Conceituais de Sowa, 1984

Um grafo conceitual é um grafo finito, conectado e bipartido. Os nós do grafo são conceitos ou são relações conceituais. Aqui não se usam arestas rotuladas, em vez disso os nós de relações conceituais representam relações entre conceitos. Como os grafos conceituais são bipartidos, os conceitos podem ter arcos apenas para relações e vice-versa. Conceitos são representados como caixas e relações conceituais como elipses.

Nos grafos conceituais os nós de conceitos (caixas) representam objetos concretos ou abstratos no mundo do discurso Exemplos: 1. O gato na cesta literal: [gato] – (na) – cesta Aqui [gato] representa uma instância de um gato, e [cesta] uma instância de cama felina. A relação (na) liga o gato à cesta.

Este conceito também pode ser representando em KIF (knowledge interchange format) através da síntaxe: (exists ((?x Cat) (?y Mat)) (On ?x ?y))

2. André vai a São Paulo de ônibus. Aqui temos 4 conceitos: [vai], [André] [São Paulo] e [ônibus]. Existem 3 relações conceituais (agente), (destino) e (instrumento). A coisa toda fica: [vai]- {o hifen informa que a definição usará mais de uma linha} (agente)–[André] (destino)–[São Paulo] (instrumento)–[ônibus]. {o . termina a definição}.

Sugestões de relacionamentos: agente, proprietário, instrumento, participante, determinante, iniciador, terminador, fonte produto, recurso, objetivo, destino, origem, demora, estado,...

 ©88-09, Pedro Kantek

10.0.5 Roteiros

Há evidências de que o ser humano represente o que sabe em estruturas correspondentes a situações típicas. Se estivermos lendo uma história sobre restaurantes, futebol ou política resolvemos qualquer ambigüidade do texto de forma consistente com restaurantes, futebol ou política. [4] Se o tema de uma história muda bruscamente, há evidências de que as pessoas interrompem brevemente a leitura, presumivelmente para trocar as estruturas de conhecimento. É difícil entender uma história pobremente organizada ou estruturada, possivelmente porque temos dificuldade em adaptá-la a uma de nossas estruturas de conhecimento pré-existentes.

Um roteiro (script) é uma representação estruturada que descreve uma seqüência estereotipada de eventos num contexto particular. O modelo de roteiro foi proposto por Schank em 1977, como um meio de organizar estruturas de dependência conceitual formando situações típicas. Os roteiros podem ser usados para organizar uma base de conhecimento em termos das situações que o sistema deve compreender.

Os componentes de um roteiro são

Condições de entrada ou descritores do mundo que devem ser verdadeiros para que o roteiro seja chamado. Por exemplo, no roteiro do restaurante, incluem-se um restaurante aberto, um cliente com fome e algum dinheiro.

Resultados ou fatos que são verdadeiros quando o roteiro tiver terminado. Neste exemplo, a fome do cliente se foi, ele está mais pobre e o dono do restaurante, mais rico.

Acessórios ou as coisas que suportam o conteúdo do roteiro. No exemplo, poderiam ser mesas, garçons, cardápio. O conjunto de acessórios permite que se faça pressuposições razoáveis sobre as situações.

Papéis Ações que os participantes individuais realizam. O garçom recebe pedidos, entrega a comida e apresenta a conta. O cliente escolhe, come e paga.

Cenas Schank divide o roteiro numa seqüência de cenas onde cada cena apresenta um aspecto temporal do roteiro. No restaurante há as cenas de entrar, pedir, comer.

Os elementos do roteiro, as peças, básicas de significado semântico, são representados usando relações de dependência conceitual. Dispostos em uma estrutura análoga a um frame eles representam uma seqüência de significados ou uma seqüência de eventos. O roteiro do restaurante retirado da pesquisa de Schank é apresentado na figura 10.7.

10.1 FASE

A FASE é um sistema construído em LISP por Guilherme Bittencourt e Augusto Costa do DAS na UFSC. Com ele podem ser construídos sistemas especialistas.

Possui os seguintes componentes:

10.1.1 Manipulador de regras

Cada regra consiste de um par de listas de padrões (esquerdo e direito). Cada padrão é uma expressão LISP podendo conter variáveis. Um padrão iterage com a base de conhecimentos através de uma lista de substituições que corresponde às instâncias presentes na base.

[4] Veja-se por exemplo a frase: Eram 46 minutos do segundo tempo, o atacante recebeu um passe primoroso e disparou um petardo com direção certa. Mas, o bandeirinha, aquele ladrão, enxergou fantasmas e ergueu sua toalha: estava cometida a suprema injustiça

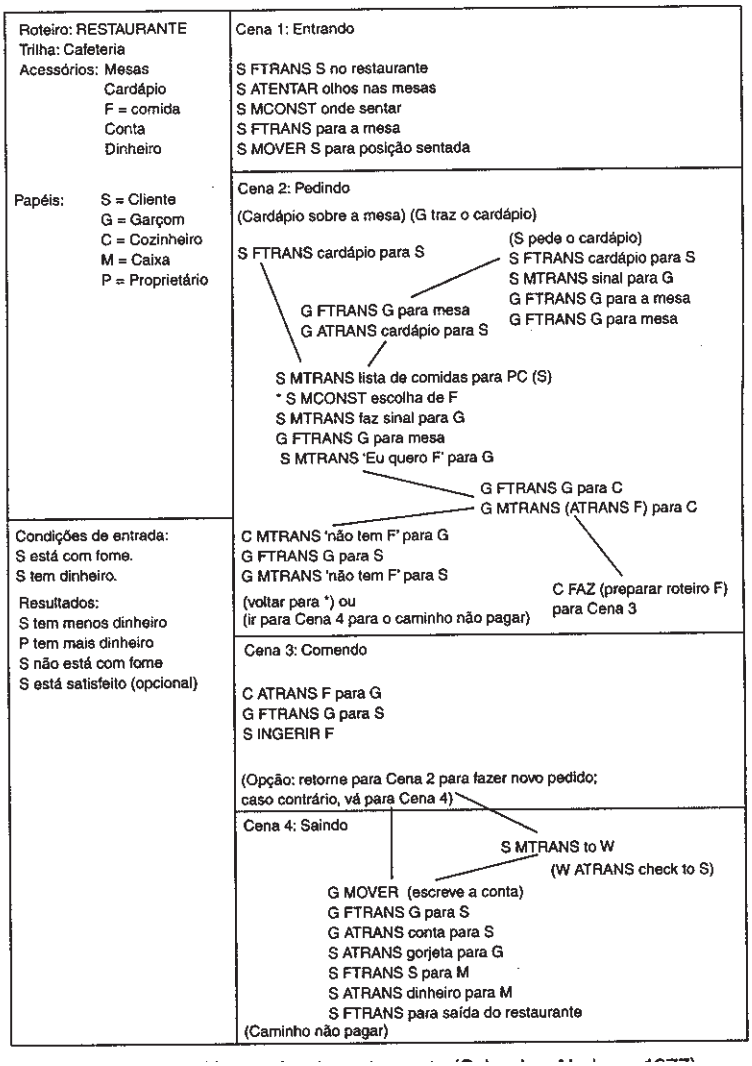


Figura 10.7: Um roteiro de uma visita ao restaurante, segundo Schank

10.1.2 Resolutor de conflitos

Como as regras serão aplicadas: alfabética do nome da regra, mas recente ou mais específica. Podem ser usadas isoladas ou combinadas

10.1.3 Representador de conhecimentos

Usa 3 formalismos distintos, compartilhando com todos um protocolo comum

- LOGICA
  Usando as regras da lógica de predicados

- Redes Semânticas (Quillian, 1968)
  Funcionam como um modelo para a memória associativa do ser humano. É um grafo orientado. Os nodos podem ser conceitos, predicados, objetos etc. Os arcos

 ©88-09, Pedro Kantek

são binários (sim/não, é-um/não-é-um). O mecanismo básico de inferência é a herança através dos arcos do grafo.

- Quadros (Minsky, 1975)
  Uma hierarquia de estruturas de dados, denominados quadros (frames). Cada frame possui atributos aos quais podem ser atribuidos valores (que podem ser ponteiros para outros quadros). Existem 3 mecanismos de inferência:
  i. Herança de valores através da hierarquia;
  ii. valores default;
  iii. chamada de uma função LISP externa ao formalismo

10.1.4 Controlador de estratégia

Existem duas: forward e backward. A primeira armazena a situação inicial na memória de trabalho. Os padrões esquerdos das regras são comparados com a memória de trabalho as regras que forem satisfeitas são selecionadas e ordenadas. Os lados direitos de tais regras geram novos fragmentos de conhecimento e o ciclo recomeça. O processamento acaba quando não houver mais regras ou após k ciclos.

Na estratégia backward, fornece-se um objetivo (goal) ao sistema, que é armazenado na memória. O sistema compara os lados direitos das regras com este objetivo, as que satisfizerem são selecionadas e ordenadas. Os lados esquerdos serão armazenados como novos objetivos.

10.1.5 Forma de uso

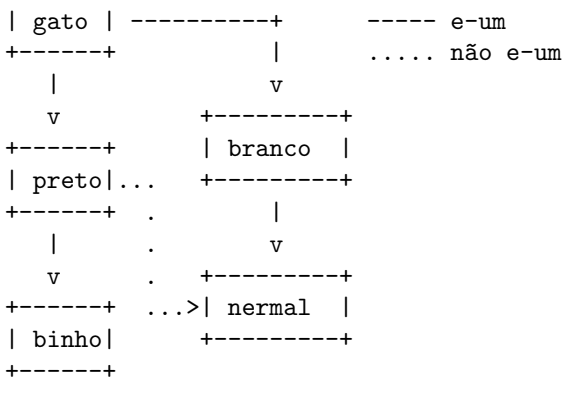
- Carregar o COMMON LISP `(c:\lisp>clisp<enter>)`
- Carregar o FASE `(?(load "all.lsp")<enter>)`
- Chamar o programa de carga de fatos, regras e objetivos

  `(?(kbase :used t)<enter>, respondendo as perguntas...)`

```
LOGICA:
Seja o seguinte arquivo PAI.F

((logic
   (pai felipe pedro)
   (pai pedro juan)))

Seja o seguinte arquivo PAI.R
(regra-1
   ((logic
      (pai ?x ?z) (pai ?z ?y)))
   ((logic
      (avo ?x ?y)))
)

Chamando o FASE, com
CLISP
(load "all.lsp")
(kbase :used t)
Arquivo de regras: pai.r <enter>
```

```
Arquivo de regras: <enter>
Arquivo de fatos: pai.f <enter>
Arquivo de fatos: <enter>
Arquivo de Objetivos: <enter>
Sistema inicializado
#S(kbase
:logic #S(kb-logic :facts ((pai pedro juan) (pai felipe pedro)))
:frame #S(kb-frame :facts nil :root #:root-1941)
:snet #S(kb-snet :facts nil)
:rules(
Regra: regra-1
Se :
(pai ?x ?z) (pai ?z ?y)
Então :
(avo ?x ?y)
) :rule-order (regra-1) :used-rules nil :asked-facts nil
:new-facts nil
:goal
s nil
:conflict-method (used) :certanly-method nil :time 0)
> (run *kb* 'for)
Início do cilco: 1
Regra: regra-1
Se :
(pai ?x ?z) (pai ?z ?y)
Então :
(avo ?x ?y)

Instâncias :
[ Substituição   :  ((?z . pedro) (?x . felipe) (?y. juan)) ]

Conclui-se que:
(avo felipe juan)

Inicio do ciclo: 2
nil
>
```

REDES SEMÂNTICAS: Seja a rede semântica



 ©88-09, Pedro Kantek

```
Seja o arquivo GATO.F
((snet (gatos (gato) (preto) (branco) (binho) (nermal)
        (gato preto e-um)
        (preto binho e-um)
        (gato branco e-um)
        (branco nermal e-um)
        (preto nermal (not . e-um)))))

Sistema inicializado
#S(kbase
:logic #S(kb-logic :facts nil)
:frame #S(kb-frame :facts nil :root #:root-3022)
:snet #S(kb-snet :facts nil)
:rules nil :rule-order nil :used-rules nil :asked-facts nil :new-facts nil
:goals nil :conflict-method (used) :certanly-method nil :time 0)
> (snet-query '(gatos (x nermal y)))
([ Substituição  : ((x . branco) (y . e-um))]
 [ Substituição  : ((x . preto) (y not . e-um))]
)

> (snet-query '(gatos (gato x y)))
([ Substituição  : ((x . binho) (y . e-um))]
 [ Substituição  : ((x . branco) (y . e-um))]
 [ Substituição  : ((x . preto) (y . e-um))]
)

> (snet-query '(gatos (x y (not . e-um))))
([ Substituição  : ((x . preto)  (y . nermal ))]
)
```

FRAMES

```
Seja o seguinte frame
((frame
  (materia root)
  (generico materia (complexidade alta) (duracao 6))
  (ed generico (duracao 12))
  (compiladores generico (complexidade media))
  (laboratorio generico (complexidade baixa))
  (arvores ed)
  (grafos ed)
  (diagramas compiladores)
  (c laboratorio)
))

Chamado (frame-list)

root-1892 <== root-1892
    materia <== root-1892
        generico <== materia
          * duracao = 6
          * complexidade = alta
```

```
            laboratorio <== generico
              * complexidade = baixa
                c <== laboratorio
            compiladores <== generico
              * complexidade = media
                diagramas <== compiladores
            ed <== generico
              * duracao = 12
                grafos <== ed
                arvores <== ed

Fazendo as seguintes perguntas
> (frame-query '(laboratorio (duracao x)))
(
[ Substituicao  :  ((x . 6)) ]
)
> (frame-query '(grafos (duracao x)))
(
[ Substituicao  : ((x . 12)) ]
)
```

Para o exemplo apresentado, construiram-se 2 bases de fatos:

```
((snet (ia (animal) (passaro) (mamifero) (pinguim) (gato) (morcego)
       (opus) (bill) (pat)
        (passaro animal e-um)
        (mamifero animal e-um)
        (pinguim passaro e-um)
        (gato mamifero e-um)
        (morcego mamifero e-um)
        (opus pinguim e-um)
        (bill gato e-um)
        (pat morcego e-um)
        ))
)

Outra:
(( frame
  (animal root)
  (generico animal (vivo t) (voa nil))
  (passaro generico (pernas 2) (voa t))
  (mamifero generico (pernas 4))
  (pinguim passaro (voa nil))
  (gato mamifero (pernas 4))
  (morcego mamifero (pernas 2) (voa t))
  (opus morcego (amigo bill))
  (bill gato (amigo opus))
  (pat morcego)
  ))
Para estas bases de fatos, podem-se fazer as perguntas:
(frame-query '(x gato)) responde bill
(frame-query '(pat (pernas x)) responde 2
(frame-query '(bill (pernas x))) responde 4
```

 ©88-09, Pedro Kantek

```
(frame-query '(pat (voa x))) responde t
(frame-query '(pat (vivo x))) responde t

Seja agora o seguinte caso:
Entrada o frame acima e a seguinte regra
(regra-1
   ((frame (?x (pernas 2))))
   ((logic (bipede ?x)))
)
e rodando-se este engenho, ele cria as seguintes entradas:
(bipede pat)
(bipede opus)
(bipede morcego)
(bipede pinguim)
(bipede passaro)

fazendo agora um
(logic-query '(bipede x))
[Substituicao  : ((x.passaro))]  From: (regra-1)
[Substituicao  : ((x.pinguim))]  From: (regra-1)
[Substituicao  : ((x.morcego))]  From: (regra-1)
[Substituicao  : ((x.opus))]  From: (regra-1)
[Substituicao  : ((x.pat))]  From: (regra-1)


Faltou especificar as regras:
(nome-regra (parte-se) (parte-então))

a parte-se e a parte-então:
(tipo expressão)...       // onde tipo é logic; snet ou frame


expressão é
(exp1) (exp2)...

Onde qualquer variavel na parte-se pode ter um ? na frente e ser
referida na parte então, quando será substituida
```

10.1.6 Tutorial no uso do FASE (UMA PARTE DO FASE...)

1. Instalar o CLISP e o FASE. Aprender a carregar o CLISP. Baixar o BAT anexo (lisp -M lispinit.mem)
O ciclo eval-apply-print. A pilha de erros e a saída (CTRL-Z).
A saida do software (exit) ou CTRL-BREAK.

2. Tentativas de familiaridade com o LISP

```
(+ 3 4)
(+ 3 4 5 6)
(zerop 6)
(not (zerop 6))
```

Qualquer coisa diferente de NIL é T. Assim zerop(0) pode ser 8 (=t)

3. Estude a função FATORIAL

```
(defun fatorial (x)
(cond ((zerop x) 1)
      (t (* x (fatorial (- x 1))))))
FAT
(fat 10)
3628800
```

4. Para carregar o FASE, fazer `(load "all.lsp") <enter>`

5. Uma possibilidade é a entrada fato a fato. Fazer

```
(logic-store '(pai felipe pedro)). Ele devolve t
(logic-store '(pai pedro juan)). Ele devolve t
(logic-list)
ele lista o conteudo dos fatos.
```

6. Outra possibilidade é criar um arquivo contendo FATOS, por exemplo PAI.F

```
((logic
 (pai felipe pedro)
 (pai pedro juan)
))
```

7. Daí, dentro do fase, deve-se chamar (kbase :used t) e responder às perguntas. Depois disso, a chamada de (logic-list) deve dar o mesmo resultado.

used t/nil → se t diz que cada regra deve ser usada 1 vez

order t/nil → se t diz que a prioridade é das regras de menor ident.

regency t/nil → se t diz que a prioridade é das regras mais recentes

special t/nil → se t diz que a prioridade deve ser das regras mais complexas

lp t/nil → com t indica uso de lógica probabilística para tratamento da incerteza

fc t/nil → com t indica fator de certeza (EMYCIN) para tratamento de incerteza

8. Formato do arquivo xxx.F (fatos) (fatos) note que o arquivo deve ser uma lista. Assim se você escrever lixo, após o ")", o lisp não dá bola...

```
fato pode ser:
 (logic (predicado termo ... ))       OU
 (logic (off (predicado termo ...))) {neste caso o predicado é anulado.}

 (snet (nodo1) (nodo2)...
       (nodo1 nodo2 aresta) ...
       [(nodo1 nodo2 (not . aresta))] )

 (frame (nome root)
       (nodo1 nome (lista1) (lista2) ... ) ... )
```

9. Exemplos

 ©88-09, Pedro Kantek

```
((snet (ia (animal) (passaro) (mamifero) (pinguim) (gato) (morcego)
       (opus) (bill) (pat)
        (passaro animal e-um)
        (mamifero animal e-um)
        (pinguim passaro e-um)
        (gato mamifero e-um)
        (morcego mamifero e-um)
        (opus pinguim e-um)
        (bill gato e-um)
        (pat morcego e-um)
        ))
)
```

Outra:

```
(( frame
  (animal root)
  (generico animal (vivo t) (voa nil))
  (passaro generico (pernas 2) (voa t))
  (mamifero generico (pernas 4))
  (pinguim passaro (voa nil))
  (gato mamifero (pernas 4))
  (morcego mamifero (pernas 2) (voa t))
  (opus morcego (amigo bill))
  (bill gato (amigo opus))
  (pat morcego)
  ))
```

10. Fazendo perguntas Anatomia de uma pergunta (logic-query '(P a b)), onde P=predicado e a,b=valor Note que a variável pode ser introduzida em qualquer posição

No exemplo do PAI, temos:

```
(logic-query '(pai x y))
x/pedro, y/juan
x/felipe, y/pedro

(logic-query '(x felipe y))
x/pai, y/pedro

(logic-query '(pai felipe x))
x/pedro
```

(snet-query '(nome-rede (origem destino arco)) Note que a variavel pode ser introduzida em qualquer posição

No exemplo da snet ia, temos

```
(snet-query '(ia (x animal e-um)))
x/pat, x/bill, x/opus, x/morcego, x/gato, x/pinguim, x/mamifero, x/passaro

(snet-query '(ia (x y (not . e-um))))
nil

(snet-query '(ia (pat morcego x)))
x/e-um
```

```
(frame-query '(x gato)) responde x/bill
(frame-query '(pat (pernas x)) responde x/2
(frame-query '(bill (pernas x))) responde x/4
(frame-query '(pat (voa x))) responde x/t
(frame-query '(pat (vivo x))) responde x/t
```

11. Regras: Anatomia de uma regra:

```
(nomeregra
 ((tipo (se) (se) ... )
   [(lisp (expressao) (expressao))] )
 ((tipo (off (não então)) (então) (então)...)))

Exemplos:
(regra1 ((logic (pai ?x ?z) (pai ?z ?y)))
        ((logic (avo ?x ?y))))

(regra2 ((frame (?x (pernas 2))))
        ((logic (bipede ?x))))

Seja agora o seguinte caso:
Entrada o frame acima e a seguinte regra
(regra-1
   ((frame (?x (pernas 2))))
   ((logic (bipede ?x)))
)
e rodando-se este engenho, ele cria as seguintes entradas:
(bipede pat)
(bipede opus)
(bipede morcego)
(bipede pinguim)
(bipede passaro)

fazendo agora um
(logic-query '(bipede x))
[Substituicao  : ((x.passaro))]  From: (regra-1)
[Substituicao  : ((x.pinguim))]  From: (regra-1)
[Substituicao  : ((x.morcego))]  From: (regra-1)
[Substituicao  : ((x.opus))]  From: (regra-1)
[Substituicao  : ((x.pat))]  From: (regra-1)
```

12. Rodando o engenho: Depois que se disse quais as regras e os fatos (e os objetivos), roda-se o engenho com (run *kb* 'for) ou (run *kb* 'back)

13. Entrada de dados:

```
(regra
 ((logic t (include (?var . (progn (princ "mensagem:" ) (read )))))
 ((logic (atributo ?var )))
)
```

Note que quando um sistema tem entrada de dados, isso inviabiliza a execução para trás (backward).

14. Uma proposta do sistema de vinhos:

 ©88-09, Pedro Kantek

```
(regra1
 ((logic t)
  (include (?p . (progn (princ "Periodo: manha/almoco/tarde/jantar/ceia: ")
                 (read)))))
 ((logic  (periodo ?p )) ))

(regra2
 ((logic t)
  (include (?c . (progn (princ "Comida:
carne/peixe/massa_leve/massa_pesada/i
ndeterminado/nada/canapes: ")
                 (read)))) )
 ((logic   (comida ?c )) ))

(regra2a
 ((logic t)
  (include (?e . (progn (princ "Quer embebedar s/n: ")
                 (read)))) )
 ((logic   (embebedar ?e )) ))

(regra3
 ((logic (periodo manha) (comida nada)))
 ((logic (vinho branco)))
)
(regra4
 ((logic (periodo manha) (comida canapes)))
 ((logic (vinho branco)))
)
(regra5
 ((logic (periodo almoco) (comida carne)))
 ((logic (vinho tinto)))
)

(regra6
 ((logic (embebedar s) (vinho tinto)))
 ((logic (tipo merlot)))
)

(regra7
 ((logic (embebedar n) (vinho tinto)))
 ((logic (tipo cabernet)))
 )

(regra8
 ((logic (embebedar s) (vinho branco)))
 ((logic (tipo chardonnay)))
 )

(regra9
 ((logic (embebedar n) (vinho branco)))
 ((logic (tipo riesling)))
 )
```

```
(regra10
 ((logic (tipo merlot)))
 ((logic (marca ma mb mc md)))
 )
(regra11
 ((logic (tipo chardonnay)))
 ((logic (marca cha chb)))
 )
(regra12
 ((logic (tipo riesling)))
 ((logic (marca ra rb rc rd)))
 )
(regra13
 ((logic (tipo cabernet)))
 ((logic (marca ca cb cc cd)))
 )
```

 ©88-09, Pedro Kantek

Capítulo 11

Visão

Os gregos supunham que os objetos enviavam réplicas de si mesmo em todas as direções sob a forma de películas finas. Tais películas, denominadas simulacra, entrariam no olho provocando a visão. Para esta teoria a visão seria um tipo de tato. Questões:

1. Como as simulacra não se atrapalham umas às outras ?
2. Como pode uma simulacra grande (de uma montanha) entrar dentro do olho ?

Outra teoria (proposta originalmente por Platão) foi a da extrusão: observando animais que enxergam bem à noite, ele propôs que os olhos emitiriam particulas de luz, que colidindo com o objeto retornariam ao olho. A crítica que Leonardo da Vinci fêz: há um tempo para ir e para voltar. Como ir até o sol ?

Kepler em 1604 matou a charada através de uma teoria adequada de refração em lentes: Kepler sugeriu que o olho serve para focar uma imagem na retina.

De qualquer maneira, ainda não se sabe se a luz é onda, partícula ou as duas coisas juntas. Esta última parece ser a teoria com mais chance de dar certo.

Segundo a visão moderna, vejamos como as coisas funcionam no nosso olho

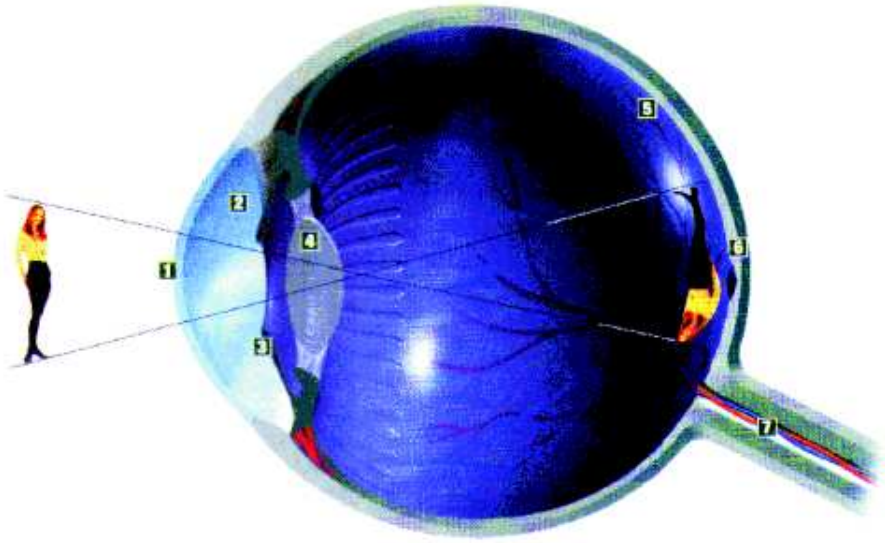


Figura 11.1: Olho humano e seus componentes

Na figura 11.1, temos as seguintes partes

1. A luz entra no olho através da córnea, que é uma lente transparente que fica na entrada do olho.

2. Aqui dentro existe o humor aquoso que mantém a pressão da córnea e age como filtro dos raios UV

3. A iris funciona como uma persiana regulando a quantidade de luz que entra. O buraco preto conhecido como pupila é isso mesmo: um buraco.

4. O cristalino, uma segunda lente faz o ajuste fino do foco

5. A retina é o elemento sensível (o filme)

6. A fóvea ou centro da retina. É o local de maior concentração de cones

7. Os axônios conduzem os sinais elétricos para o cérebro.

O globo ocular tem cerca de 20mm de diâmetro. A pupila varia entre entre 2 e 8 mm. As células que identificam as imagens são de dois tipos: cones e bastonetes. Há 7 milhões de cones e cerca de 120 milhões de bastonetes.

Cones e Bastonetes

Os cones reconhecem as 3 cores fundamentais (azul, verde e vermelho), enquanto os bastonetes conseguem reconhecer apenas a presença de luz, logo eles vêm em branco e preto. Os cones são muito mais sensíveis a baixos níveis de luz, daí "à noite, todos os gatos são pardos".

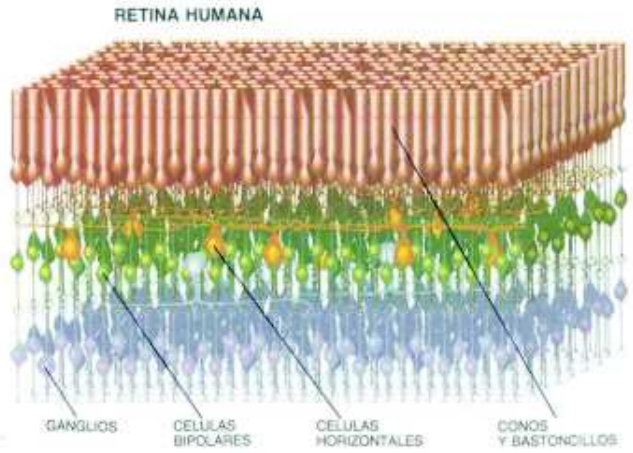


Figura 11.2: Cones e bastonetes na retina

O olho está adaptado para funcionar em uma enorme faixa de valores de intensidade luminosa, na ordem de 10^{10}, embora ele não possa atuar nesta faixa simultaneamente. O olho transita nessa faixa, adaptando-se, no fenômeno conhecido como adaptação de brilho. Uma manifestação deste fato está na saída da matineé, quando ao deixar uma sala escura e sair na claridade da tarde, nosso olho reclama da claridade intensa e rapidamente se adapta a ela.

Alguns fatos

- Em tempo claro, uma vela é vista a 16 Km
- 10^{-14} da energia da vela é suficiente para estimular a visão
- 120.000.000 de bastonetes/olho
- 6.000.000 cones/olho
- 1.000.000 terminações nervosas/olho
- 250.000.000 receptores versus 250.000 em uma TV
- bastonetes são 500 vezes mais sensíveis à luz
- Um bastonete responde a uma ativação de um único quanta, (limite físico máximo de excitação
- intervalo de intensidade 10^{16} (160 dB)
- o olho não tem obturador: o cérebro captura o fluxo ótico de maneira ininterrupta

11.1 Cérebro

O elementos de processamento do cérebro é o neurônio. Conhecem-se cerca de 100 tipos de neurônios, embora não se conheça muito bem a diferença funcional entre eles. Eis alguns dados sobre o cérebro:

- número de elementos: 10.000.000.000 a 1.000.000.000.000 neurônios
- tamanho / volume: volume no homem: 1500 cc
- diâmetro do neurônio: cerca de 0,004 polegadas.
- comprimento máximo do axônio: cerca de 1 m.
- peso: 1450 gramas
- potência elétrica: 10 Watts
- velocidade de transmissão e chaveamento: de 10 a 100 m/seg. A mielina aumenta em 20×. Chaveamento máximo: 500/Seg
- interconexão de neurônios: 200.000 conexões / neurônio (nas células de Purkinje). Estas conexões são conhecidas como sinapses.
- confiabilidade: Confiabilidade do neurônio: baixa (neurônios morrem continuamente). Confiabilidade do sistema: alta.
- expectativa de funcionamento: cerca de 70 anos.
- codificação da informação: digital, com modulação em freqüência.

 ©88-09, Pedro Kantek

11.1.1 Rodopsina

A transformação de impulsos visuais em elétricos, se dá nas células cones e nas bastonetes através da ação de uma proteina chamada rodopsina. Veja-se a seguir uma descrição do funcionamento da bacteriorodopsina, similar à aquela, encontrada na membrana de uma bactéria, mas mais estável e com melhores propriedades óticas.

Na bacteriorodopsina, ocorrem fotociclos. O fotociclo é uma resposta ao estímulo da luz: Em repouso, a molécula está num estado que foi chamado bR. Recebendo um feixe de luz verde, em alguns trilhonésimos de segundo, a molécula vai ao estado K que logo depois relaxa para o estado M e este ao estado O. Aqui, a molécula espera alguns milissegundos e se nada acontecer, a molécula relaxa para bR novamente. Entretanto, se no estado O, durante a espera, a molécula for atingida por luz vermelha, ela decai ao estado P que é instável e rapidamente vai para Q que tem estabilidade de alguns anos. Só sai daqui pela ação da luz azul que leva novamente a molécula ao estado bR. O nome deste processo é "arquitetura seqüencial mono-fotônica", e é graças a este complexo processo é que enxergamos cores.

Esta molécula tem sido usada para a construção de bits em biocomputadores. O estado bR é lido como bit 0 e qualquer estado intermediário é usado como bit 1. (Maiores detalhes em "Computadores baseados em proteínas, BIRGE, Robert em Scientific American, março de 1995).

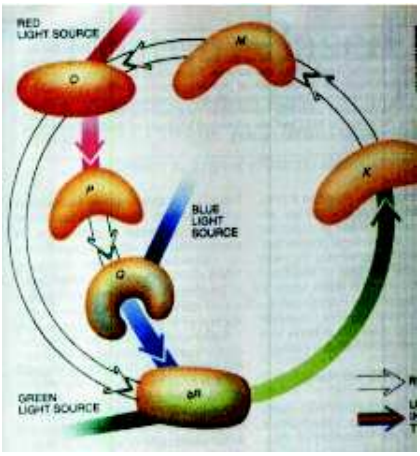


Figura 11.3: O ciclo da rodopsina no cérebro

11.1.2 Uma visão bioquímica do ciclo

Quando a luz atinge a retina, um fóton interage com uma molécula denominada 11-cis-retinal, que se rearranja em um pico-segundo (um trilionésimo de segundo) e se transforma em transretinal. (Um picosegundo é mais ou menos o tempo que a luz leva para cruzar a largura de um único cabelo humano). A mudança na forma da molécula retinal força uma mudança na forma da proteína, a rodopsina, à qual a retinal está fortemente ligada. A metamorfose da proteína altera seu comportamento. Nesse momento denominada de metarrodopsina II, a proteína cola-se a outra proteína, chamada transducina. Antes de transformar-se em metarrodopsina II, a transducina liga-se fortemente a uma pequena molécula chamada GDP. Mas, quando a transducina interage com a metarrodopsina II, a GDP se desprende e uma molécula chamada GTP cola-se à transducina. (A GTP mantém uma estreita relação com a GDP, mas é muito diferente dela.)

A GTP-transducina-metarrodopsina II liga-se agora a uma proteína chamada fosfodiesterase, localizada na membrana interna da célula. Quando ligada à metarrodopsina II e a seu grupo, a fosfodiesterase adquire a capacidade única de "cortar"uma molécula chamada CGMP (um elemento químico aparentado a GDP e GTP). Inicialmente, há grande número de moléculas CGMP na célula, mas a fosfodiesterase reduz sua concentração da mesma maneira que a retirada da tampa do ralo baixa o nível de água em uma banheira.

Outra membrana de proteína que se liga à CGMP é denominada de canal iônico. Ela funciona como um portão que regula o número de íons de sódio na célula. Normalmente, o canal permite que íons de sódio entrem na célula, enquanto uma proteína separada os bombeia ativamente para fora. A ação dupla do canal iônico e da bomba mantém o nível de íons de sódio na célula dentro de uma faixa estreita. Quando o volume de CGMP é reduzido devido à divisão efetuada pela fosfodiesterase, o canal iônico se fecha, fazendo com que seja reduzida a concentração celular de íons de sódio positivamente carregados. Esse fato ocasiona um desequilíbrio de carga de um lado a outro da membrana da célula que, enfim, faz com que uma corrente seja transmitida pelo nervo óptico até o cérebro. O resultado, quando interpretado pelo cérebro, é a visão.

Se as reações mencionadas acima fossem as únicas que ocorrem na célula, o suprimento de 11-cis-retinal, CGMP e íons de sódio logo seria esgotado. Alguma coisa tem que desativar as proteínas que foram ligadas e fazer a célula voltar a seu estado inicial. Vários mecanismos se encarregam disso. Em primeiro lugar, no escuro, o canal iônico (além dos íons de sódio) também deixa que íons de cálcio penetrem na célula. O cálcio é bombeado para fora por uma proteína diferente, de modo a ser mantida uma concentração constante de cálcio. Quando os níveis de CGMP caem, fechando o canal iônico, decresce também a concentração de íons de cálcio. A enzima fosfodiesterase, que destrói a CGMP, diminui em uma concentração mais baixa de cálcio. Em segundo, uma proteína denominada guanilato ciclase começa a ressintetizar a CGMP quando os níveis de cálcio começam a cair. Em terceiro, enquanto tudo isso acontece, a metarrodopsina II é quimicamente modificada por uma enzima chamada rodopsina cinase. A rodopsina modificada liga-se a uma proteína conhecida como arrestina, que impede que a rodopsina ative mais transducina. A célula, portanto, contém mecanismos que limitam o sinal amplificado iniciado por um único fóton. A transretinal por fim desprende-se da rodopsina e precisa ser reconvertida em 11-cis-retinal e, mais uma vez, ligada à rodopsina para voltar ao ponto de partida de outro ciclo visual. A fim de realizar isso, a transretinal primeiro é modificada quimicamente por uma enzima e transformada em transretinol - uma forma que contém mais dois átomos de hidrogênio. Uma segunda enzima converte, em seguida, a molécula em 11-cis-retinol. Enfim, uma terceira enzima remove os átomos de hidrogênio previamente acrescentados a fim de formar o 11-cis-retinol, e o ciclo se completa. (Michael J. Behe, A Caixa Preta de Darwin, págs. 28 a 30).

11.2 Psicologia da visão

Olho ou cérebro ? Costuma-se pensar que o orgão de visão é o olho, mas tendemos a nos esquecer que o principal órgão de visão é o cérebro, atuando o olho como sensor e o cérebro como interpretador. Afinal a imagem que o olho recebe é um pontilhado discreto. Parece um quadro de Seurat. Atente-se para a eficiência deste sistema: se 10 fotons atingirem um único cone, você verá luz. Como você vê pontos discretos, mas enxerga linhas contínuas, sobra um trabalho de construção no meio. Tal trabalho é feito por partes do cortex cerebral, e embora ele não seja de todo conhecido já há sérias evidências sobre seu funcionamento.

 ©88-09, Pedro Kantek

Figura 11.4: Um quadro do pintor Seurat

11.2.1 Inteligência Visual

A maneira moderna de entender a inteligência visual afirma: "o cérebro CONSTRÓI as imagens do que nós vemos"Por exemplo, veja-se a onda. Esta figura é bi ou tridimensional ? Claramente o desenho é bidimensional, mas quase todas as pessoas percebem a profundidade subjacente. Idem para o quadrado mágico [Hof00, pág.3]

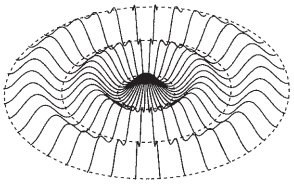


Figura 11.5: Iso é bi ou tri-dimensional ?

Figura 11.6: Consegue enxergar o quadrado ?

Existem diversas maneiras de "enxergar"a visão. Buscando inspiração nos filósofos, temos:

- Sentido fenomenal: ver as coisas "como lhe parecem"
- Sentido relacional: a coisa deve EXISTIR para ser vista no sentido relacional (vide filme mente brilhante)

Isto nos leva ao *Problema fundamental da visão: A imagem no olho tem infinitas interpretações possíveis.*

Estas interpretações são aprendidas pela criança (citar o caso de Cheselden [Hof00, pág 17] Veja-se por exemplo, o cubo de Necker, na figura 11.2.1. Questões: 1. Quando vejo A, onde está B ? 2. Eu vejo A e você vê B. Quem está certo ?

Problema fundamental da profunidade: A imagem no olho tem 2 dimensões. Logo tem incontáveis interpretações em 3D

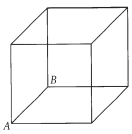


Figura 11.7: Cubo de Necker

Você não constrói profundidade a seu bel prazer, e sim usando REGRAS. O objetivo delas é minimizar ou eliminar a ambigüidade.

Regra1: construa apenas mundos para os quais a imagem é uma visão estável (Ou seja mudando um pouco a cabeça, ela continua igual). Se mudar, temos um ACIDENTE DE PONTO DE VISTA"

Por exemplo se seu olho vê um Vezão (um V grande) tende a imaginar que as duas hastes estão conectadas na parte inferior. Uma outra possibilidade, é a de que as hastes estivessem separadas por 1 metro e apenas o posicionamento delas e de seu olho permitisse enxergá-las como juntas. O olho sempre prefere a primeira hipótese.

Sub-regras: 2D -> 3D: 1. Interprete uma linha reta em 2D como reta em 3D 2. Se as extremidades de 2 retas coincidem em 2D, elas coincidem em 3D 3. Linhas colineares em uma imagem serão colineares em 3D 4. elementos próximos na imagem estarão próximos em 3D (exemplo fig. pag. 31) 5. regra da projeção (fig. pág 37) 6. quando for possível interprete uma curva em uma imagem como um aro de uma superfície 3D 7. O conceito da junção em T: uma junção T em uma imagem é o ponto onde o aro completo se esconde; o topo esconde a haste. 3D -> partes: Divida formas em partes ao longo de dobras côncavas

11.2.2 Ilusões de ótica

A seguir, algumas ilusões de ótica

Figura 11.8: Ilusão de ótica

Figura 11.9: Ilusão de ótica

11.3 Estereogramas

Um estereograma é uma imagem que "engana"o cérebro forçando-o a construir algum objeto ou desenho que não está diretamente na imagem. Para explicar o conceito de estereograma, vejamos um mais simples, construído apenas com caracteres.

 ©88-09, Pedro Kantek

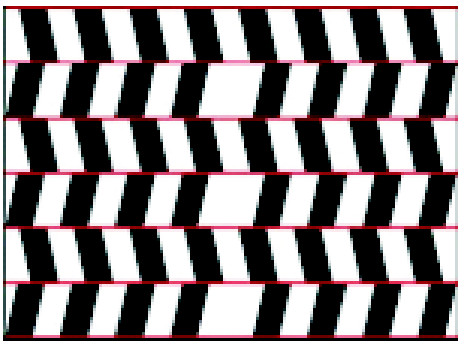


Figura 11.10: Ilusão de ótica

Figura 11.11: Ilusão de ótica

Para enxergar um estereograma a questão chave é acostumar o olho a focar uns 20 cm atrás da imagem real. Forçando o olho a isto, cada um dos dois olhos verá uma imagem diferente. Uma das maneiras do cérebro corrigir esta discrepância (imagens diferentes !) é supor que o objeto que ele (o cérebro) está vendo está construído em níveis.

Começaremos desenhando qualquer coisa como sendo uma repetição de padrões. Por exemplo

```
12345678901234567890123456789012345678901234567890...
```

Fazendo uma pequena alteração no padrão, ficamos com:

```
12345678901234567890123456789123456789123456789123456789...
```

Note que a partir do um sublinhado, o padrão se modificou, o zero deixou de fazer parte dele. Ao fazer o foco lá atrás, o olho esquerdo vê um zero, enquanto o olho direito não o vê (porque ele foi tirado!). Uma maneira do cérebro corrigir esta anomalia é supor a existência de um degrau no ponto onde o zero sumiu:

```
1234567890123456789012345678
                            12345678912345678912345678...
```

Pensando assim, o cérebro corrige a anomalia: o olho esquerdo vê o zero e o degrau, enquanto o olho direito tem o zero coberto pelo degrau. Se quisermos retornar ao nivel anterior, basta inserir o zero em algum lugar à direita do ponto original e fica:

```
1234567890123456789012345678                    012345678901234567890...
                            123456789123456789
```

O stereograma acima seria assim escrito em uma única linha

```
12345678901234567890123456789123456789123456789012345678901234567890...
```

Olhando a linha acima, o cérebro veria (verá) um degrau no meio da linha.

O estereograma acima é de uma única linha. Mas ele pode ser criado com quantas linhas quisermos, e o detalhe interessante é que cada uma das linhas pode ter um padrão diferente. A única regra a respeitar é:

a) Deve haver um padrão de reptição na linha b) Para subir um nivel, exclua um elemento do padrão de repetição c) Para descer um nivel, inclua um elemento no padrão (não precisa ser aquele que foi excluido).

Retorne agora ao exemplo acima (o do melhor time do Paraná) e localize a subida e a descida de nivel.

Um software disponível na Internet para brincar com estereogramas é o POPPRO.EXE. Localize-o via Google ou Yahoo e boa sorte.

Um exemplo de estereograma de caracteres

```
PBGNJQRUVDFLMTSWPBGNJQRUVDFLMTSWPBGNJQRUVDFLMTSWPBGNJQRUVDFLMTSWPBGNJQRUV
LVTCSWNYHGKMAFRPLVTCSWNYHGKMAFRPLVTCSWNYHGKMAFRPLVTCSWNYHGKMAFRPLVTCSWNYH
FSZNQAOBJWXEKCHYFSZNQAOBJWXEKCHYFSZNQAOBJWXEKCHYFSZNQAOBJWXEKCHYFSZNQAOBJ
UMYFZVLAJKOQTXWUMYFZVOLAJKOQTXUMYFZVOLAJKOQTXUMQYFZVOLAKOQTXUMQYEFZVOLAKO
KPMIQSFWURHDATYKPMIQSFWUJRHDATKPMIQSFWUJRHDATKPHMIQSFUJRHDATKPHMIQSDFUJRH
PUHTLYRODJXIVGBPUHTLYRODJXJIVGPUHTLYRODJXJIVGPUXHTLRODJXJIVGPUXHTLRODIJXJ
FUGLIZOHCPATJDPWFUGLIOHCPATAJDWFUGLITOHCPATAJDWFUGITOHCPAJTAJDWFGITOHCDPA
VJCGWKMPLXRAFXUQVJCGWKMPLXRAFXQVJCGWRKMPLXRAFXQVJGWRKMPLAXRAFXQVJGWRKMPLA
ZDYCRPQSVWLXWFHEZDYCRPQSVWLXWFEZDYCRLPQSVWLXWFEZDCRLPQSXVWLXWFEZDCRLPQSXV
SPBVMIRCQKZEKJGFSPBVMIRCQKZEKJFSPBVMIRCQKZZEKJFSPVMIRCQEKZZEKJFSPVMIRCQEK
DBUNSJIEQFAKFOMVDBUNSJIEQFAKFOVDBUNSJIEQFAAKFOVDBNSJIEQKFAAKFOVDBNSJIEQKF
CRWZISAGETVMQTPKCRWZISGEVTVMQTKCRWZISGEVTVMMQTKCRZISGEVTQVMMQTKCRISPGEVTQ
SBQCZKEFYJULRJPTSBQCZEFYJULURJTSBQCZLEFYJULURJTSBQZLEFYJRULURJTSQZLEFYPJR
AOFBCIYHQLKDGNTLAOFCIYHQLKDKGNLAOFCIDYHQLKDKGNLAOFCDYHQLKDGKGNAOFCDYHQNLK
FZAKRYCBGTLEWIJFZAKRYCBGTTLEWIFZAKRYLCBGTTLEWIFZAKRYCBGTTLEWIFZAKRYCEBGTT
UWJVDZLPRAXOQGSUWJVDZLPXRAXOQGUWJVDZILPXRAXOQGUWJVDZILXRAXOQGUWJVDOZILXRA
BSWTUCKELMXIQNOYSWTXUCKELMXIQNYSWTXUICKELMXIQNYSWTXUICKELMXQNYQSWTXUICKEL
SNYWRQIPDZJHXMBFSNYWRQIPDZJHXMBFSNYWRQIPDZJHXMBFSNYWRQIPDZJHXMBFSNYWRQIPD
UJIMNLSVBCPOEHTXUJIMNLSVBCPOEHTXUJIMNLSVBCPOEHTXUJIMNLSVBCPOEHTXUJIMNLSVB
GHWFULYTSVDMJZKBGHWFULYTSVDMJZKBGHWFULYTSVDMJZKBGHWFULYTSVDMJZKBGHWFULYTS
```

Desafio

- Implementar este algoritmo de reconstituição de imagens.
- Construir um programa que gere estereogramas randômicos para arquivos de texto.
- Manipular diretamente arquivos BMP, principalmente em seu bloco de controle, observando os efeitos das manipulações. Diversão certa!.

11.4 Tomografia computadorizada axial

Em 1862 um ginasiano alemão com 17 anos, de nome Guilherme Roentgen, foi apanhado em uma daquelas armadilhas do destino: presenciara um colega desenhar uma caricatura ofensiva do diretor do ginásio. Note-se que o desenho era ofensivo no ponto de vista do diretor. Na visão dos alunos era o maior barato. O diretor, quando viu o desenho, embraveceu-se, subiu nas tamancas: quem cometera aquela ofensa ? Ninguém se apresentou, mas por azar, o diretor soube que Guilherme sabia quem era o autor: Chamado às falas, teve diante de si uma escolha: ou dedurar o colega ou manter-se calado. Escolheu a segunda opção e foi expulso da escola. Da escola e do país, pois ao ser excluído do colégio não seria aceito por nenhum outro colégio alemão, que a alemãozada é fogo! Que remédio, o negócio foi se mudar para a Suíssa e lá começar o curso de engenharia mecânica. Na Escola Politécnica de Zurique ele foi bem apagado, mas as rodas do destino estão sempre lá: Trabalhava nessa mesma escola, o Dr August Kundt, um dos mais famosos físicos teóricos da Europa. Kundt encantou-se por uma habilidade de Roentgen: embora fosse uma aluno meio medíocre, tinha uma capacidade imensa de produzir aparelhos físicos. Que funcionavam. Com madeira, vidro e algumas chapas de metal era capaz de construir qualquer engenhoca digna do professor Pardal. Começaram a trabalhar juntos e juntos seguiram por 3 universidades. Cada vez que Kundt era

 ©88-09, Pedro Kantek

convidado para um novo posto, Roentgen o acompanhava. Aos 34 anos, já doutor em física teórica, este último abandonou o mestre e assumiu uma cátedra na Universidade de Giessen. Já era físico até a raiz dos cabelos, esquecera-se da engenharia. Ele fazia medidas de radiações e uma de suas ferramentas era um tubo de Crookes, inventado em 1861 na Inglaterra, e que vinha a ser uma lampadona com cátodo e ânodo e no qual se fazia vácuo. No seu trabalho, usava fotografias, que nessa época eram embaladas uma a uma em estojos de madeira para sua proteção contra a luz. Um dia, deixou algumas dessas fotos virgens e seladas próximas ao tubo de Crookes enquanto trabalhava e quando foi usar as chapas em fotografias comuns, percebeu na revelação, um monte de manchas. Reclamou do fabricante dizendo que ele lhe mandara fotos meio veladas. O fabricante retrucou dizendo que as chapas fotográficas estavam perfeitas quando sairam da fábrica. O que Roentgen não sabia (mas já já ia descobrir) era que o tubo de Crookes gerava uma radiação invisível, que tinha a capacidade de ultrapassar madeira e que velava chapas fotográficas. Para quem não percebeu, está-se a falar de uma radiação que quando descoberta foi batizada por Roentgen de "Raios-X", sendo este "X"usado no mesmo sentido de "desconhecido", tal como quando dizemos em uma equação matemática x igual a alguma coisa. No início foi uma mera curiosidade científica: um dispositivo que podia ser usado para fotografar coisas atrás de barreiras. Roentgen começou a estudar quais materiais bloqueavam os RX e percebeu que o melhor deles era o chumbo: uma fina película de chumbo impedia a passagem dos raios. Em algum momento de dezembro de 1895, Roentgen segurava um cachimbo próximo a uma chapa fotográfica bombardeada por RX. Ao revelá-la, percebeu assustado que além do cachimbo uma mão de caveira segurava o dito cujo: a mão era a dele, e ele estava enxergando os próprios ossos. Imediatamente percebeu o alcance daquela descoberta: Agora seria possível olhar para dentro do corpo sem precisar usar facas e serras.

Querendo convencer o mundo, convidou a mulher Bertha Roentgen – que até então nada sabia – a segurar uma chapa não velada, que foi devidamente bombardeada. Quando a chapa foi revelada, estava lá a mão da mulher e no dedo anular um anel. Foi pela joia que a mulher se convenceu que era a sua mão com os seus ossos. Veja esta foto de 1895 mais abaixo, no texto. O cientista esperava uma reação de júbilo e alegria, mas deu zebra, foi um anticlímax: a mulher ficou enfurecida, achou uma invasão de privacidade o marido olhar assim, "dentro"dela. Aliás, nos primeiros anos dos RX, a reação foi a pior possível, onde já se viu devassar as entranhas das pessoas.

Em pouco tempo, coisa de meses, a novidade se espalhou, Roentgen foi aclamado como um benfeitor na humanidade . Ele ganhou o primeiro Nobel de física em 1901. Fêz duas coisas que nunca mais alguém teve a coragem de fazer: Recebeu o prêmio em silêncio, sem discursar já que era tímido e deixou o valor do prêmio para a Universidade onde trabalhava. Com isto encerra-se o primeiro capítulo desta nossa história: o surgimento do RaioX. Quem quer que tenha passado 15 minutos na sala de espera da clínica de Fraturas do alto da XV pode bem avaliar o que significou esta descoberta para o nosso dia-a-dia.

A tomografia computadorizada foi inventada pelo engenheiro eletrônico Godfrey N. Hounsfield (1919-), pela qual recebeu o prêmio Nobel em fisiologia e medicina em 1979, juntamente com o sul-africano naturalizado americano, físico Allan McLeod Cormack (1924-). Cormack desenvolveu em 1956 a teoria e a matemática de como múltiplos raios projetados sobre o corpo, em ângulos diferentes, mas em um único plano, forneceriam uma imagem melhor do que o raio único, usado na radiografia. Seus trabalhos foram publicados no Journal of Applied Physics, em 1963 e 1964. Sendo este um jornal de física, não era lido por radiologistas, e Hounsfield desenvolveu a teoria independentemente, e construiu o aparelho.

Foi Housnfield que batizou o processo que acabava de inventar com o nome pomposo de tomografia computadorizada axial transversa. Vem a ser uma técnica para reconstruir

imagens bi-dimensionais de seções transversais de pacientes a partir de um conjunto de fluxos de RX unidimensionais. As vantagens de tal facilidade são óbvias: ao invés de examinar vagas sombras em um fotograma de Raio-X convencional, médicos podem examinar alterações patológicas na anatomia com o mesmo grau de claridade que teriam se tivessem cortado o paciente em 2. Embora pareça uma máquina de RaioX, não se enganem: o que tem lá dentro é um computador. Até tem um RX, mas este é assessório ao computador. E, onde há computador, há programas e onde há programas, há algoritmos: pronto. Estamos na nossa praia, demorou mas chegou.

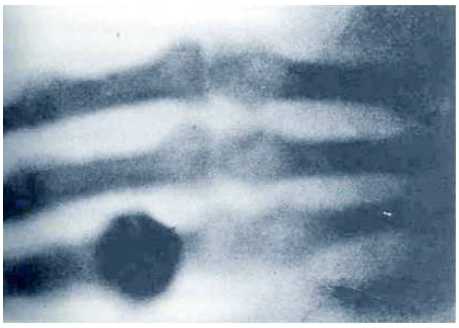


Figura 11.12: A radiografia de Roengten

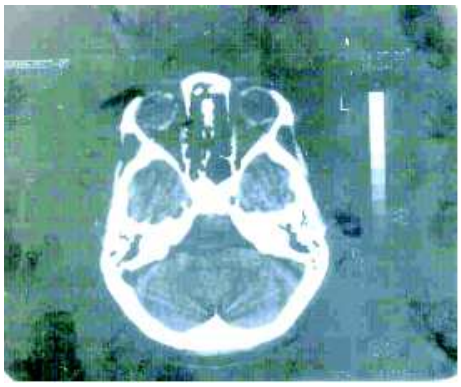


Figura 11.13: Uma tomografia de cérebro

Coeficiente de Atenuação O coeficiente de atenuação linear média μ_t de cada pixel é comparado com o coeficiente da água, μ_a, definindo o número CT:

$CT = 1000(\mu_t - \mu_a)/\mu_a$.

A água é utilizada como referência porque seu coeficiente de atenuação é similar ao dos tecidos moles, e é um material fácil de obter para calibrar os aparelhos. O coeficiente 1000 é utilizado para obter números inteiros.

O número CT, ou coeficiente de Hounsfield, é definido como -1000 para o ar e 0 para a água. Para os tecidos em geral, ele depende da energia do feixe empregado. Por exemplo, para 80 keV, se o coeficiente de atenuação linear típico de ossos é de 0,38 cm^{-1}, e da água 0,19 cm^{-1}, o número CT dos ossos é de +1000. Pode ser ainda maior para ossos corticais. Estes valores também variam de aparelho para aparelho, já que os coeficientes dependem da distribuição de energia do feixe. A radiação observada, I, está relacionada com a radiação na fonte, I_0, por:

$I = I_0 e^{-\mu x}$.

Tecido	CT
Ar	-1000
Pulmão	-900 a -400
Gordura	-110 a -65
Água	0
Rim	30
Sangue normal	35 a 55
Sangue coagulado	80
Músculo	40 a 60
Fígado	50 a 85
Ossos	130 a 250

Por convenção, altos valores de CT são imageados como branco, e baixos como preto.

 ©88-09, Pedro Kantek

Como o olho humano não pode distinguir os milhares de coeficientes, utilizamos a técnica de janelas (windowing), para graficar somente os valores em uma certa faixa.

Aquisição de dados Na tomografia computadorizada mais comum, um tubo com um feixe de cerca de 0,6 mm de diâmetro gira em torno do paciente, e emerge do paciente sobre um detector com aproximadamente 700 sensores, que convertem a intensidade em uma corrente. Cada pulso de raio-X dura 2 a 3 ms, completando uma volta em cerca de 1 s. Cada 360° gera 300 somas.

Cada vez que o tubo emite um pulso, cada detector mede o logaritmo da intensidade que recebe. Este valor representa a soma de todos os números CT dos voxeis atravessados pelo raio, completando uma projeção. Cada voxel é atravessado pelo feixe em diferentes direções, durante a rotação do anel. O número CT de cada voxel está portanto representado em várias somas. (Estas últimas informações foram retiradas em http://www.if.ufrgs.br/ast/med/imagens/imagens.htm por Kepler de Souza Oliveira Filho).

Algoritmo de reconstrução

O algoritmo de reconstrução é fácil de colocar e entender. Para fazer isto vamos pensar em uma sessão transversal de um corpo humano. Imagine também um conjunto de emissores de raio-X, colocados paralos em uma estrutura rígida que gira em um eixo estabelecido no centro do corpo humano. No lado oposto (ultrapassado o corpo) estão os detectores do raio-X.

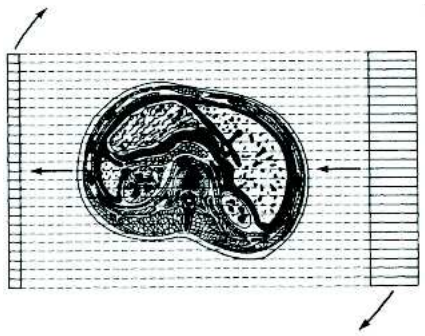


Figura 11.14: Funcionamento do tomógrafo

À medida em que a geringonça gira, inumeros conjuntos de Raios-X vão sendo tomados. Para uma determinada posição angular (digamos horizontal) tem-se um conjunto de x valores de atenuação. O valor x é a quantidade de emissores e detetores, e a atenuação é a diminuição da potência do Raio-X emitido numa ponta e recebido na outra após passar pelas estruturas do corpo que está sendo estudado.

Por exemplo, na horizontal, os raios inferiores, tem que atravessar a coluna (supondo que o paciente esteja deitado de costas), ao passo que os raios superiores ou não atravessam nada, ou apenas alguma gordura (isso para os mais gordinhos). Quando a estrutura estiver tirando raios X na vertical, a coluna do paciente atenuará os raios do meio do conjunto.

Os detectores não tiram fotografias, ao invés eles mandam sinais elétricos proporcionais a potência do Raio-X recebido diretamente para um computador. Este, de posse da informação referente ao ângulo de tomada e dos valores de todos os sensores, vai reconstituindo do corpo que está sendo "cortado".

Vamos exemplificar o caso, usando uma instância bem simples do problema. Seja uma caixa retangular que contém um cubo sólido dentro dela. Não se conhecem as dimensões nem a posição do cubo. Fazendo 3 tomadas a 0, 45 e 90 graus, como se vê na figura 11.15.

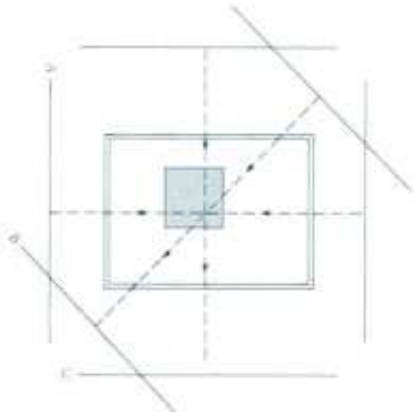


Figura 11.15: tomografia com 3 tomadas

São feitas 3 tomadas de RX. A primeira, representada por (A), atravessa a caixa no sentido longitudinal e sensibiliza sensores colocados após a caixa em A. A segunda, atravessa a caixa fazendo 45 graus com as arestas da caixa e sensibiliza os sensores em B. Finalmente o terceiro fluxo atravessa a caixa de maneira transversal.

Supondo que tanto a caixa quanto o objeto cúbico absorvem algum RX, e supondo um emissor linear percorrendo toda a área de sensoreamento, paralelo aos eixos A, B e C, poderíamos ter as seguintes leituras nos sensores:

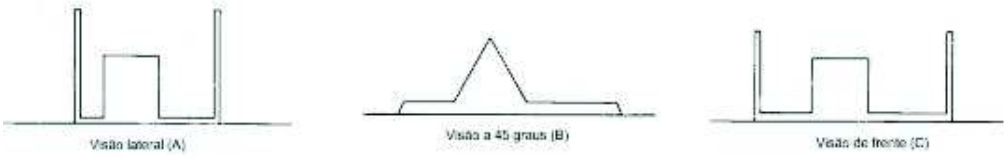


Figura 11.16: as 3 leituras em 11.15

Note-se que na visão lateral (A), os primeiros e últimos detetores (bem à esquerda e bem a direita) não são atenuados, chegam com a mesma potência com que saíram, isto é não atravessaram nenhuma estrutura. Note que na visão a 45 graus, quando o fluxo começa a cruzar a caixa ele cresce, chega a um máximo (que corresponde à diagonal do cubo) e em seguida diminui no mesmo ritmo.

A técnica mais simples de reconstrução é a chamada "back projection", e todas as demais que vieram depois são melhoramentos e variações desta.

Esta técnica tem uma formulação matemática bem compacta:

$$P(\theta, t) = \int_R f(x, y)\delta(x \sin\theta - y \cos\theta - t)dxdy$$

Essa fórmula pode ser lida como: P é a atenuação medida no sensor t, posicionado em um ângulo θ em relação ao eixo horizontal. x, y são as coordenadas dos pontos da

 ©88-09, Pedro Kantek

imagem e δ é um filtro que tem a função de só considerar os pontos que estão no caminho do fluxo t de Raios X.

Esclarecendo melhor a função de δ, é uma função simples que vale 1 quando o ponto x, y está no caminho do fluxo t e vale 0 quando não está. Para quem se lembra um pouco de geometria analítica, esta formula já deve ter sido vista. De fato, $x \sin\theta - y \cos\theta = t$ vem a ser a equação paramétrica da reta que atravessa os eixos no ponto t e com inclinação θ.

Como o que se tem é a atenuação (medida pelos sensores) e o que se quer obter é o mapa de todos os x, y (ou seja a imagem do corpo que está sendo "cortado"), há que se tomar a função inversa aquela acima e tem-se:

$$f(x,y) = \frac{1}{2\pi} \int_0^{\pi} \int_{-r}^{+r} p(\theta,t)\delta(x\sin\theta - y\cos\theta - t)dtd\theta$$

O ponto x,y da imagem é igual à somatória das atenuações, obtidas em todas as medidas angulares (variação de θ), para todos os fluxos de RaiosX (variação de t), mas apenas para os pontos que estão no caminho do fluxo t (filtro θ).

Para encerrar o capítulo matemático desta história, basta converter a fórmula acima para o mundo real. Não podemos nos esquecer que o computador não é uma máquina analógica, contínua. Ele é digital, que neste sentido é sinônimo de discreta. Assim, as integrais precisam ser convertidas em somatórios, e a formula final fica:

$$f(x_i y_j) = \frac{1}{n} \sum_{k=1}^{n} \sum_{l=1}^{m} p(\theta_k, t_l)\delta(x_i \sin\theta_k - y_j \cos\theta_k - t_l)$$

Escrevendo isso na forma bem mais amigável de um programa, e supondo que queiramos reconstruir uma imagem de LIN por COL pixels, a partir de tomogramas individuais temos:

```
1: função RECONSTOMOGR (atenuações P)
2: inteiro i, j, k, l
3: para (i = 1; i ≤ LIN; i++) faça
4:    para (j = 1; j ≤ COL; j++) faça
5:       IMAGEM[i; j] ← 0
6:    fimpara
7: fimpara
8: para (k = 1; k ≤ n; k++) faça
9:    para (l = 1; l ≤ m; l++) faça
10:      para (i = 1; i ≤ LIN; i++) faça
11:         para (j = 1; j ≤ COL; j++) faça
12:            IMAGEM[i; j] ← IMAGEM[i; j] + P(θ_k, t_l).d(x_i, y_j, θ_k, t_l)
13:         fimpara
14:      fimpara
15:   fimpara
16: fimpara
17: para (i = 1; i ≤ LIN; i++) faça
18:    para (j = 1; j ≤ COL; j++) faça
19:       IMAGEM[i; j] ← (1/nm).IMAGEM[i; j]
20:    fimpara
21: fimpara
22: imprima IMAGEM
```

Para concluir este passeio, vamos tentar reconstruir um objeto 2D, colocado no meio de uma treliça 11 x 11. Nós estamos vendo o objeto de cima, mas a tomografia se fará apenas a partir de atenuações laterais. O objeto vai ser representado por 11 níveis de

"tinta". O nível 0, indica que não há nada aí, e niveis crescentes indicam que o objeto é cada vez mais espesso nesse lugar Correspondendo a seguinte distribuição de valores

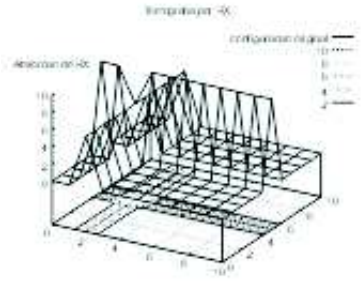


Figura 11.17: Um corpo sendo tomografado

```
   .00   .00  6.00   .00   .00   .00   .00   .00   .00   .00   .00
   .00   .00  6.00   .00   .00   .00   .00   .00   .00   .00   .00
   .00   .00  6.00   .00   .00   .00   .00   .00   .00   .00   .00
   .00   .00  6.00   .00   .00   .00   .00   .00   .00   .00   .00
   .00   .00  6.00   .00   .00   .00   .00   .00   .00   .00   .00
 10.00 10.00  6.00 10.00 10.00 10.00 10.00 10.00 10.00 10.00 10.00
   .00   .00  6.00   .00   .00   .00   .00   .00   .00   .00   .00
   .00   .00  6.00   .00   .00   .00   .00   .00   .00   .00   .00
   .00   .00  6.00   .00   .00   .00   .00   .00   .00   .00   .00
   .00   .00  6.00   .00   .00   .00   .00   .00   .00   .00   .00
   .00   .00  6.00   .00   .00   .00   .00   .00   .00   .00   .00
```

Trata-se de uma cruz de material, sendo a haste horizontal de material que absorve mais o RX (representado por 10) e a haste vertical na coluna 3, de um material mais "leve", já que está representado apenas por 6. Os demais pontos não contém nada (valor 0).

O resultado da tomografia em 2 angulos (0 e 90), apresenta 11 níveis de absorção para cada ângulo, e gera a seguinte configuração

```
  .00   .55   .55   .55   .55   .55  9.64   .55   .55   .55   .55   .55
90.00   .91   .91  6.00   .91   .91   .91   .91   .91   .91   .91   .91
```

E o reconstituição da configuração original a partir desta tomografia é
com a seguinte configuração

```
   .73   .73  3.27   .73   .73   .73   .73   .73   .73   .73   .73
   .73   .73  3.27   .73   .73   .73   .73   .73   .73   .73   .73
   .73   .73  3.27   .73   .73   .73   .73   .73   .73   .73   .73
   .73   .73  3.27   .73   .73   .73   .73   .73   .73   .73   .73
   .73   .73  3.27   .73   .73   .73   .73   .73   .73   .73   .73
  5.27  5.27  7.82  5.27  5.27  5.27  5.27  5.27  5.27  5.27  5.27
   .73   .73  3.27   .73   .73   .73   .73   .73   .73   .73   .73
   .73   .73  3.27   .73   .73   .73   .73   .73   .73   .73   .73
   .73   .73  3.27   .73   .73   .73   .73   .73   .73   .73   .73
   .73   .73  3.27   .73   .73   .73   .73   .73   .73   .73   .73
   .73   .73  3.27   .73   .73   .73   .73   .73   .73   .73   .73
```

 ©88-09, Pedro Kantek

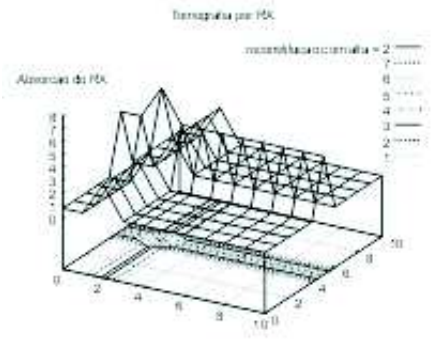


Figura 11.18: resultado da tomografia de 11.17

Note que o desenho é quase igual (já que os ângulos coincidiram com as hastes da cruz: esperteza nossa :-)...), mas observando a matriz numérica de reconstituição percebem-se valores diferentes de 6 e 10, embora nas mesmas posições da figura original.

Agora, fazendo a tomografia e a reconstituicao com 10 angulos (0, 18, 36, 54, 72, 90, 108, 126, 144 e 162 graus) tem-se como resultado

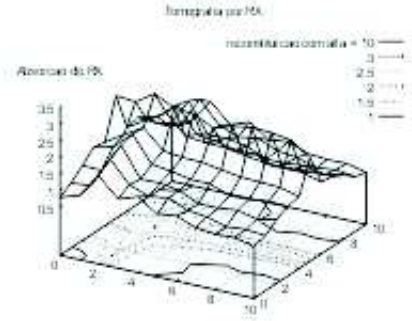


Figura 11.19: Reconstituição da imagem original

Com a seguinte configuração

```
 .00   .00   .45   .45  1.36  1.36  1.73   .36  1.09   .00   .00
 .00   .00   .45   .45  1.36  1.36  1.73   .36  1.09   .00   .00
 .45   .45   .91   .91  1.82  1.82  2.18   .82  1.55   .45   .45
 .45   .45   .91   .91  1.82  1.82  2.18   .82  1.55   .45   .45
1.36  1.36  1.82  1.82  2.73  2.73  3.09  1.73  2.45  1.36  1.36
1.36  1.36  1.82  1.82  2.73  2.73  3.09  1.73  2.45  1.36  1.36
1.73  1.73  2.18  2.18  3.09  3.09  3.45  2.09  2.82  1.73  1.73
1.09  1.09  1.55  1.55  2.45  2.45  2.82  1.45  2.18  1.09  1.09
 .36   .36   .82   .82  1.73  1.73  2.09   .73  1.45   .36   .36
 .00   .00   .45   .45  1.36  1.36  1.73   .36  1.09   .00   .00
 .00   .00   .45   .45  1.36  1.36  1.73   .36  1.09   .00   .00
```

Para concluir, pode-se relembrar que este caso acima listado é apenas didático. A resolução 11x11 é muito pobre, e as imagens não sofreram nenhum tratamento (transformadas de Fourier, limiarização, regularização), mas ele aponta bem para a simplicidade

e poder do conceito original. Da próxima vez que você se vir dentro daquele tubo que gira fazendo barulho, enquanto sua cabeça é esquadrinhada, fique com medo apenas do que poderá aparecer lá dentro. A máquina não deve assustar mais.

11.4.1 Outras técnicas de reconstituição por imagem

PET-Tomografia por emissão de positrons Aqui o paciente recebe um traçador metabolicamente ativo. Moléculas carregando isótopos, que tenham afinidade com a área do corpo a ser pesquisada. (Exemplo: a glucose rotulada com C^{11} acumula no cérebro onde é usada como fonte primária de energia. Tem meia vida de 20 minutos). O isótopo radioativo decai emitindo um positron. Este colide com um eletron antes de andar 1mm. A iteração de ambas gera 2 raios γ que emergem em sentidos opostos e são detectados em volta do paciente, por dois detectores. A colisão terá se dado na linha que une os dois detectores, salvo se tiver havido coincidência acidental (causa de ruído). Depois de 500.000 aniquilações a imagem é reconstituída. Esta é a técnica ideal para pesquisa médica pela sua não invasividade e qualidade.

Resonância Nuclear Os elementos possuem quase todos isótopos magnéticos. Quando o corpo é imerso em um campo magnético, uma pequena fração de núcleos se alinha com este campo (Um alinhamento de 1 em um milhão já é mensurável). Se o campo é alternado a uma freqüência de ressonância apropriada, os spins nucleares mudam de orientação. Ao mudar, eles absorvem energia. Quando o campo é desligado, os núcleos retornam ao estado de equilíbrio emitindo energia na mesma freqüência que receberam. Os núcleons de diferentes elementos, e mesmo de diferentes isótopos do mesmo elemento, têm frequências de ressonância diferentes. Para um campo de 0,1 T (1000 Gauss), a frequência de ressonância dos prótons é de 4,2 MHz, e a do fósforo é de 1,7 MHz.

Mamografia A xero-radiografia, desenvolvida por Wolf e Ruzicka em 1956, emprega o mesmo princípio de uma foto-copiadora. Ao invés de filme, o cassete contém uma folha rígida de alumínio sobre o qual é formada uma fina camada de selênio. Antes da exposição ao raio-X, a superfície é coberta (spray) com uma carga positiva uniforme. No escuro, o material é isolante, mas quando exposto ao raio-X (ou luz), os elétrons são liberados dentro do material, e progressivamente descarregam a carga positiva da superfície. A carga remanescente é proporcional á exposição ao raio-X submetida. Portanto, o padrão de raio-X foi convertido a um padrão de cargas positivas. A imagem então é processada em um aerosol de fina partículas de carbono ou toner termo-plástico, carregadas negativamente, que são atraídas pelas cargas positivas. A imagem é então transferida a uma folha de papel que é aquecida para fixar as partículas do toner permanentemente. A placa de selênio pode então ser limpa e re-usada. O toner se acumula nas partes onde houve maior atenuação de raio-X, e portanto é o reverso de um raio-X normal. A imagem geralmente tem baixo contraste, mas estruturas de cerca de 1 mm sobressaem bastante.

11.5 imagens mapeadas

Uma imagem monocromática pode ser descrita por uma função f(x,y), que determina, no ponto x,y a quantidade de brilho (ou o nivel de cinza) no ponto. f(x,y) representa o produto da iluminância i(x,y) que incide sobre o ponto pela refletância r(x,y) que indica qual a percentagem de luz incidente nesse ponto é refletida. Matematicamente

$$f(x, y) = i(x, y) \times r(x, y),$$

 ©88-09, Pedro Kantek

Figura 11.20: Reconstituição do exemplo da folha de exercícios: etapa 1

Figura 11.21: Reconstituição do exemplo da folha de exercícios: etapa 2

com $0 \leq i \leq \infty$ e $0 \leq r \leq 1$.

Eis alguns valores consagrados para i em lux ou lúmen/m2

Dia ensolarado	900 lux
dia nubaldo	100 lux
iluminação de um escritório	10 lux
noite de lua cheia	0,001

Eis também os valores de r

Neve	0,93
parede recém pintada de branco fosco	0,80
aço inoxidável	0,65
veludo preto	0,01

A captura de uma imagem é realizada através de uma digitalização obtida através de um equipamento sensor (câmera, scanner,...) Este processo também implica em uma discretização, que será no espaço (amostragem) e na amplitude (quantização).

O resultado final é uma matriz de L linhas por C colunas contendo N níveis de brilho, que usualmente são chamados de níveis de cinza, já que o valor mais alto representa o

Figura 11.22: Reconstituição do exemplo da folha de exercícios: etapa 3

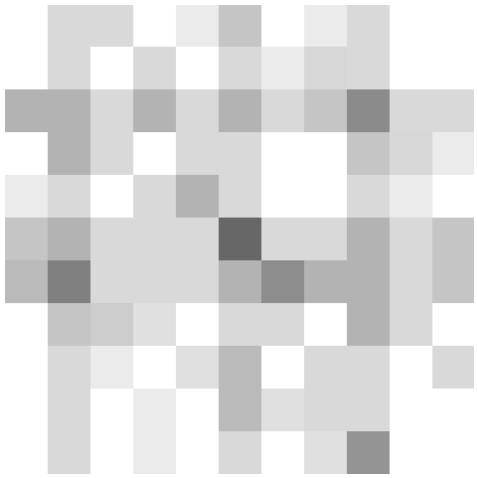


Figura 11.23: Reconstituição do exemplo da folha de exercícios: etapa 4

branco e o mais baixo, o preto. Cada um dos números desta matriz é chamado de pixel, contração de picture element.

As quantidades L, C e N são um compromisso entre tamanho e qualidade da imagem versus a quantidade de memória necessária para guardar a imagem.

11.6 Arquivos BMP

O padrão BMP (bit mapping) é um dos mais simples padrões de representacao de imagens no ambiente windows. Grosso modo, uma imagem BMP é um arquivo que contém as seguintes informações:

- Bloco de controle
- Tabela cores (se a imagem for mapeada)
- Imagem

 ©88-09, Pedro Kantek

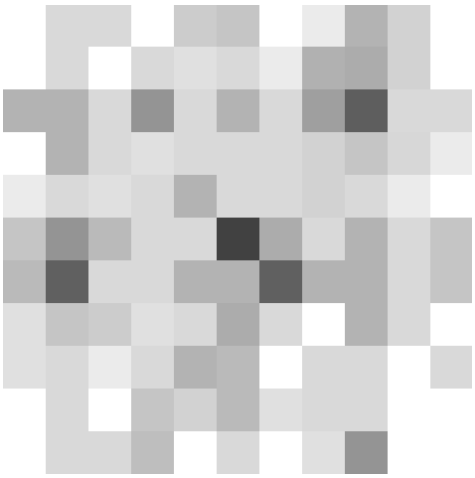


Figura 11.24: Reconstituição do exemplo da folha de exercícios: etapa 5

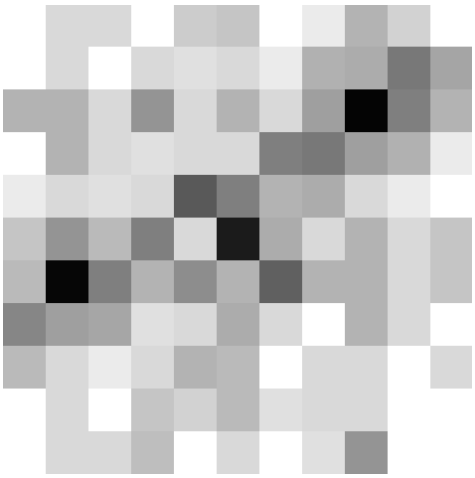


Figura 11.25: Reconstituição do exemplo da folha de exercícios: etapa 6

O bloco de controle tem a seguinte configuração

Tipo	Tam	Desl	Nome	Descrição	exemplo
char	2	0	type	constante igual a BM	424D
long int	4	2	size	tamanho do arquivo	C6 04 00 00
int	2	6	reserved	zeros	00 00
int	2	8	reserved	zeros	00 00
long int	4	A	off	desloc até a imagem	36 04 00 00
long int	4	E	size	tamanho bloco controle	28 00 00 00
long int	4	12	width	n. de colunas	0A 00 00 00
long int	4	16	height	n. de linhas	0C 00 00 00
int	2	1A	planes	planos	01 00
int	2	1C	bitcount	bits / pixel	08 00
long int	4	1E	compress	compressao	00 00 00 00
long int	4	22	sizeimage	tamanho da imagem	90 00 00 00
long int	4	26	pix/m H	pixels / m na horizontal	CE 0E 00 00
long	4	2A	pix/m	na vertical	D8 0E 00

Note que

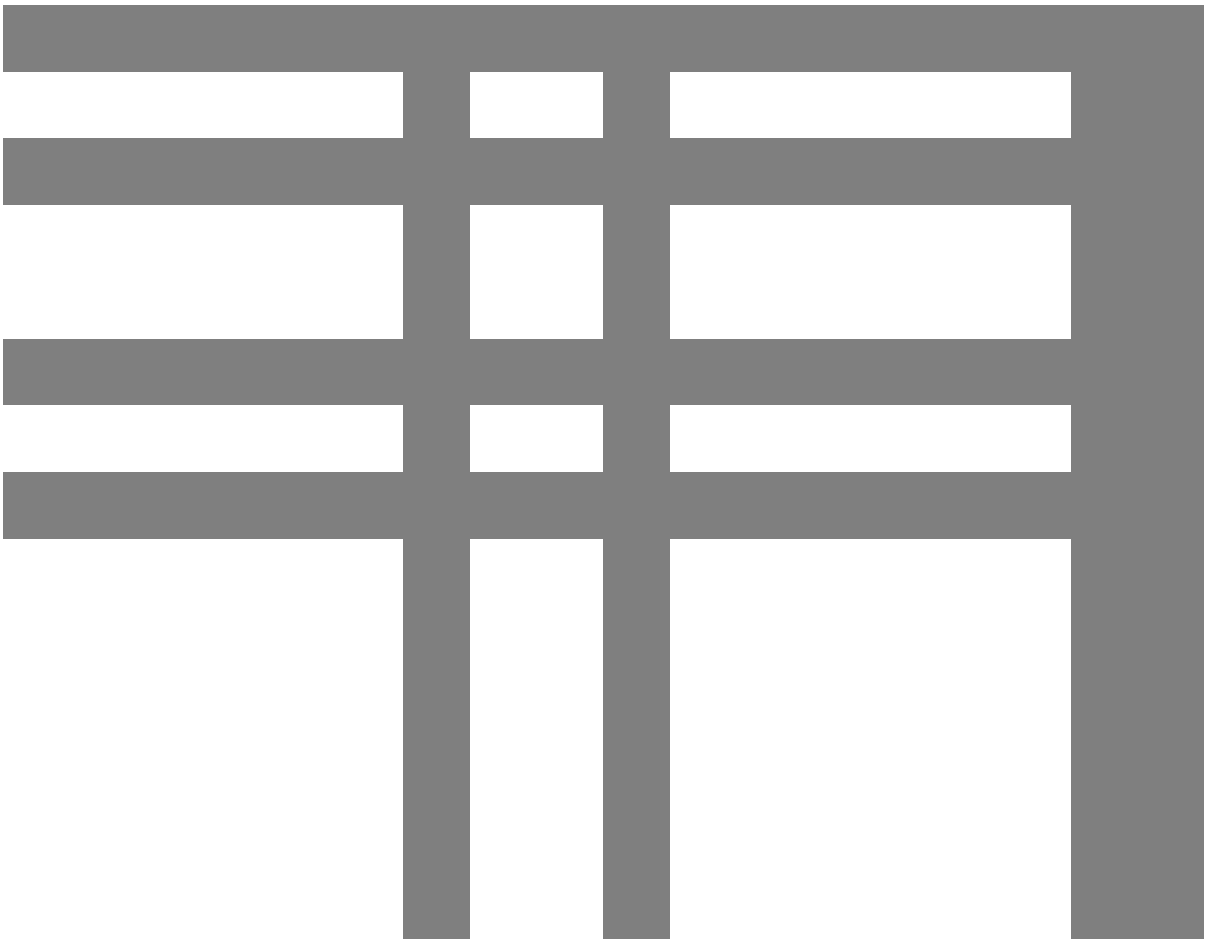
- A tabela de cores tem tamanho variável (1024 bytes no máximo), começa no endereço 36 e cada entrada contém as quantidades de azul/verde e vermelho. (note a inversão)
- O quarto byte de cada entrada é sempre zeros binários.
- A imagem começa em 536 (se a tabela = 1024) e cada linha está alinhada em múltiplo de 32 bits, por questões de eficiência.
- Na tabela, cada número indica a quantidade de tinta, assim: FF = branco e 00 = preto.

Seja como exemplo uma imagem devidamente *dumpeada*: É uma imagem branca, contendo 4 linhas V e 4 linhas H de cor cinza.

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
     -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
0000  42 4D 62 05 00 00 00 00 00 00 36 04 00 00 28 00 BM........6...(.
0010  00 00 13 00 00 00 0F 00 00 00 01 00 08 00 00 00 ................
0020  00 00 60 01 00 00 00 00 00 00 00 00 00 00 00 01 ................
0030  00 00 00 00 00 00 00 00 00 00 01 01 01 00 02 02 ................
0040  02 00 03 03 03 00 04 04 04 00 05 05 05 00 06 06 ................
0050  06 00 07 07 07 00 08 08 08 00 09 09 09 00 0A 0A ................
0060  0A 00 0B 0B 0B 00 0C 0C 0C 00 0D 0D 0D 00 0E 0E ................
0070  0E 00 0F 0F 0F 00 10 10 10 00 11 11 11 00 12 12 ................
0080  12 00 13 13 13 00 14 14 14 00 15 15 15 00 16 16 ................
...
0410  F6 00 F7 F7 F7 00 F8 F8 F8 00 F9 F9 F9 00 FA FA ................
0420  FA 00 FB FB FB 00 FC FC FC 00 FD FD FD 00 FE FE ................
0430  FE 00 FF FF FF 00 FF FF FF FF FF FF 7F FF FF 7F ................
0440  FF FF FF FF FF FF 7F 7F FF 00 FF FF FF FF FF FF ................
0450  7F FF FF 7F FF FF FF FF FF FF 7F 7F FF 00 FF FF ................
0460  FF FF FF FF 7F FF FF 7F FF FF FF FF FF FF 7F 7F ................
0470  FF 00 FF FF FF FF FF FF 7F FF FF FF 7F FF FF FF ................
0480  FF FF 7F 7F FF 00 FF FF FF FF FF FF 7F FF FF 7F ................
0490  FF FF FF FF FF FF 7F 7F FF 00 FF FF FF FF FF FF ................
04A0  7F FF FF 7F FF FF FF FF FF FF 7F 7F FF 00 7F 7F ................
04B0  7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
04C0  7F 00 FF FF FF FF FF FF 7F FF FF 7F FF FF FF FF ................
04D0  FF FF 7F 7F FF 00 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
04E0  7F 7F 7F 7F 7F 7F 7F 7F 7F 00 FF FF FF FF FF FF ................
04F0  7F FF FF 7F FF FF FF FF FF FF 7F 7F FF 00 FF FF ................
0500  FF FF FF FF 7F FF FF 7F FF FF FF FF FF FF 7F 7F ................
0510  FF 00 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
0520  7F 7F 7F 7F 7F 00 FF FF FF FF FF FF 7F FF FF 7F ................
0530  FF FF FF FF FF FF 7F 7F FF 00 7F 7F 7F 7F 7F 7F ................
0540  7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 00 FF FF ................
0550  FF FF FF FF 7F FF FF 7F FF FF FF FF FF FF 7F 7F ................
0560  FF 00                                          ..
```

A imagem tem 15 linhas e 19 colunas. Os riscos estão nas linhas 7, 9, 12 e 14 e nas colunas 7, 10, 17 e 18.

 ©88-09, Pedro Kantek



EXERCÍCIO 36 Você verá abaixo as principais partes de um arquivo grafico BMP. Trata-se de uma imagem retangular, branca, contendo 4 linhas verticais e 4 linhas horizontais de cor cinza.

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
     -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

0000  42 4D 8A 05 00 00 00 00 00 00 36 04 00 00 28 00 BM........6...(.
0010  00 00 12 00 00 00 11 00 00 00 01 00 08 00 00 00 ................
0020  00 00 60 01 00 00 00 00 00 00 00 00 00 00 00 01 ................
0030  00 00 00 00 00 00 00 00 00 00 01 01 01 00 02 02 ................
0040  02 00 03 03 03 00 04 04 04 00 05 05 05 00 06 06 ................
0050  06 00 07 07 07 00 08 08 08 00 09 09 09 00 0A 0A ................
0060  0A 00 0B 0B 0B 00 0C 0C 0C 00 0D 0D 0D 00 0E 0E ................
0070  0E 00 0F 0F 0F 00 10 10 10 00 11 11 11 00 12 12 ................
...
0410  F6 00 F7 F7 F7 00 F8 F8 F8 00 F9 F9 F9 00 FA FA ................
0420  FA 00 FB FB FB 00 FC FC FC 00 FD FD FD 00 FE FE ................
0430  FE 00 FF FF FF 00 FF 7F FF FF FF FF 7F FF 7F FF ................
0440  FF FF FF FF FF FF 7F FF 00 00 FF 7F FF FF FF FF ................
0450  7F FF 7F FF FF FF FF FF FF FF 7F FF 00 00 FF 7F ................
0460  FF FF FF FF 7F FF 7F FF FF FF FF FF FF FF 7F FF ................
```

```
0470  00 00 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
0480  7F 7F 7F 7F 00 00 FF 7F FF FF FF FF 7F FF 7F FF ................
0490  FF FF FF FF FF FF 7F FF 00 00 FF 7F FF FF FF FF ................
04A0  7F FF 7F FF FF FF FF FF FF FF 7F FF 00 00 FF 7F ................
04B0  FF FF FF FF 7F FF 7F FF FF FF FF FF FF FF 7F FF ................
04C0  00 00 FF 7F FF FF FF FF 7F FF 7F FF FF FF FF FF ................
04D0  FF FF 7F FF 00 00 FF 7F FF FF FF FF 7F FF 7F FF ................
04E0  FF FF FF FF FF FF 7F FF 00 00 FF 7F FF FF FF FF ................
04F0  7F FF 7F FF FF FF FF FF FF FF 7F FF 00 00 FF 7F ................
0500  FF FF FF FF 7F FF 7F FF FF FF FF FF FF FF 7F FF ................
0510  00 00 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
0520  7F 7F 7F 7F 00 00 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
0530  7F 7F 7F 7F 7F 7F 7F 7F 00 00 FF 7F FF FF FF FF ................
0540  7F FF 7F FF FF FF FF FF FF FF FF 7F FF 00 00 7F ................
0550  7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
0560  00 00 FF 7F FF FF FF FF 7F FF 7F FF FF FF FF FF ................
0570  FF FF 7F FF 00 00 FF 7F FF FF FF FF 7F FF 7F FF ................
0580  FF FF FF FF FF FF 7F FF 00 00                   ..........
```

Pergunta-se:
Quantas linhas tem a imagem ? _____ e quantas colunas ? _____ os riscos cinza estão nas linhas _____, _____, _____ e _____. e nas colunas _____, , ._____ e _____.

Desafio

- Construir um programa que gere estereogramas randômicos para arquivos de texto.
- Manipular diretamente arquivos BMP, principalmente em seu bloco de controle, observando os efeitos das manipulações. Diversão certa!.

11.7 Modificação de histograma

O histograma de uma imagem com 256 níveis de cinza é um vetor de 256 inteiros contendo a quantidade de pixels da imagem que tem um determinado nivel de cinza. Por exemplo, supondo que a imagem M tenha 100×100 pixels seu histograma seria o vetor V contendo 256 inteiros. A primeira inteiro ($V[1]$) conteria a quantidade de pixels da imagem com nivel 0 (preto). $V[2]$ conteria a quantidade de pixels com nível 1, e assim por diante, até o elemento $V[256]$ que teria a quantidade de pixels totalmente brancos, ou seja com nível=255. Neste caso, se se somar todos os valores existentes nas 256 posições de V deve-se chegar a 10.000, que vem a ser a quantidade total de pixels da imagem. Eis outro exemplo mais simples. Seja a imagem M_1 contendo 6 linhas por 10 colunas com 16 níveis de cinza (de 0 a 15).

$$M_1 = \begin{bmatrix} 1 & 12 & 13 & 11 & 1 & 1 & 0 & 0 & 0 & 0 \\ 11 & 10 & 11 & 11 & 11 & 0 & 0 & 0 & 0 & 0 \\ 10 & 13 & 13 & 13 & 11 & 11 & 0 & 0 & 0 & 0 \\ 11 & 11 & 10 & 6 & 5 & 1 & 0 & 0 & 0 & 0 \\ 11 & 10 & 12 & 11 & 1 & 1 & 0 & 0 & 0 & 0 \\ 11 & 11 & 11 & 11 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

 ©88-09, Pedro Kantek

Neste caso, o histograma de M_1 é $V = (25, 8, 0, 0, 0, 1, 1, 0, 0, 0, 4, 15, 2, 4, 0, 0)$. O mesmo deve ser interpretado afirmando que há 25 pixels pretos, 8 pixels valendo 1, ... 15 pixels valendo 11, e assim por diante.)

11.8 Escalonamento de histograma

Outra técnica utilizada visando reduzir a quantidade de níveis de cinza de uma determinada imagem é a redução da variedade de níveis de cinza a uma gama mais estreita de valores. Por exemplo a passagem de uma imagem de 256 níveis de cinza para outra com 16 níveis de cinza pode significar uma economia de espaço da ordem de 50%. Isto ocorre porque são necessários 8 bits para armazenar um binário entre 0 e 255 e apenas 4 bits para uma número binário entre 0 e 15. Eis o algoritmo:

1: função ESCALONAMENTO (IMAGEM [LIN] [COL] de inteiros)
2: inteiro I, J
3: **para** $(I = 1;\ I \leq LIN;\ I++)$ **faça**
4: **para** $(J = 1;\ J \leq COL;\ J++)$ **faça**
5: $IMAGEM[I][J] \leftarrow \lfloor IMAGEM[I][J] \div 16$
6: **fimpara**
7: **fimpara**
8: devolva IMAGEM

Um exemplo de transformação de uma imagem de 256 níveis de cinza em apenas 16.

Figura 11.26: Figura com 256 níveis de cinza

Figura 11.27: A figura 11.26 com apenas 16 níveis

11.8.1 Equalização de histograma

Esta transformação visa distribuir os níveis de cinza pelo histograma de maneira a aumentar o contraste da imagem. A grande maioria dos programas de edição de imagem possui esta função em geral sob um nome parecido a "contraste automático"ou similar. Dentro da transformação há que se ter uma função que governe a redistribuição dos niveis de cinza. Diversas funções podem ser usadas, mas a mais comum é a função de distribuição acumulada. É esta a que será estudada aqui. Seja o algoritmo:

1: função EQUALIZHIST (inteiro IMAGEM [LIN] [COL])

```
2: inteiro I, J
3: real MAT[256][5]
4: para (I = 1; I ≤ LIN; I++) faça
5:    para (J = 1; J ≤ COL; J++) faça
6:       MAT[IMAGEM[I][J]][1]++
7:    fimpara
8: fimpara
9: para (I = 1; I ≤ 255; I++) faça
10:   MAT[I][2] ← MAT[I][2] ÷ 255
11: fimpara
12: para (I = 1; I ≤ 255; I++) faça
13:   MAT[I][3] ← MAT[I][1] ÷ (LIN × COL)
14: fimpara
15: MAT[1][4] ← MAT[1][3]
16: para (I = 2; I ≤ 255; I++) faça
17:   MAT[I][4] ← MAT[I − 1][3] + MAT[I][3]
18: fimpara
19: para (I = 1; I ≤ 255; I++) faça
20:   MAT[I][5] ←"o índice da linha de MAT na qual o
21:   valor da coluna [2] MAT está mais perto MAT[I][4]"
22: fimpara
23: para (I = 1; I ≤ LIN; I++) faça
24:   para (J = 1; J ≤ COL; J++) faça
25:      IMAGEM[I][J] ← MAT[IMAGEM[I][J]][5]
26:   fimpara
27: fimpara
28: devolva IMAGEM
```

Acompanhe no exemplo. A imagem 11.28 contém uma pequena amplitude valores de tinta. O pixel mais escuro corresponde a 50 e o mais claro a 72. A amplitude de cor da imagem é portanto de apenas 22.

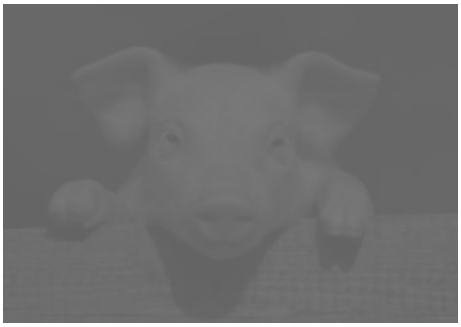


Figura 11.28: Figura com pouco contraste

Figura 11.29: A figura 11.28 devidamente equalizada

11.8.2 Distribuição linear de níveis de cinza

Este algoritmo é parecido com o anterior, no sentido de que ele busca aumentar o contraste. A idéia é descobrir qual a cor piso (a mais baixa na imagem) e depois qual a cor teto (a mais alta). Conhecendo estes dois valores, cabe fazer uma equivalencia onde o piso passa a corresponder a 0 e o teto a 255, interpolando-se todos os demais valores

 ©88-09, Pedro Kantek

neste intervalo. O algoritmo não será mostrado pois é bastante simples. Ficar-se-á apenas com um exemplo. Suponha-se a existência da imagem M_2 contendo um número qualquer de pixels. Na construção do histograma, verificou-se que existem apenas 3 valores de cinza na imagem. Digamos que seja uma imagem com pouco contraste, e na qual há apenas as cores 60, 70 e 80. Logo, a imagem é quase preta, com pouquissima variação. Aplicar a distribuição lienar de cinza equivale a aumentar a amplitude original (que é de apenas $80 - 60 = 20$) para 255. Com isso, os pixels que eram 60 passam a ser 0. Os que eram 70 passam a 127 e os que eram 80 passam a 255. Transformou-se a imagem de cinza quase homogêneo para uma figura onde há preto (0), cinza (127) e branco(255). O nome "linear" vem do fato de que não importa quanto de cada cor havia na imagem original, para efetuar a transformação para a nova cor. Note-se que na equalização de histograma a informação do peso de cada cor na imagem é relevante e é considerada na distribuição.

11.8.3 Compressão de histograma

Esta transformação busca diminuir o contraste da imagem. Ela pode ser programada de diversas maneiras, mas em geral deve-se dividir o valor de cada pixel por uma determinada constante. Por exemplo, se uma determinada imagem tiver seus pixels divididos por 10, ela passará a ter apenas 10% do contraste original.

11.8.4 Limiarização

A limiarização consiste em escolher um valor limite (*o limiar*) e depois transformar todo pixel inferior ou igual a este limite em zero (preto) e todo pixel maior do que este limite em 255 (branco). Após uma operação de limiarização, a imagem pode passar a ser representada por uma matriz binária e não mais matriz de inteiros. Em termos práticos a imagem pode passar a ter seus pixels armazenados em bits e não mais em bytes, causando uma economia significativa de espaço. Quanto ao valor a empregar como limite de separação, existem diversas abordagens. As duas mais usuais são o valor do meio do intervalo, que é o 127 e que corresponderia a um cinza mediano. Outro valor também empregado, é o nível de cinza k, tal que a metade dos pixels da imagem fique branca e a outra metade fique preta. Esta limiarização preserva um certo equilibrio de tom na imagem, mas tem duas desvantagens: Primeiro o valor tem que ser calculado imagem a imagem, e segundo, varia de uma imagem a outra dificultando o trabalho conjunto de diversas imagens. Eis o algoritmo:

```
1: função LIMIARIZAÇÃO (IMAGEM [LIN] [COL] de inteiros, k inteiro)
2: inteiro I, J
3: para (I = 1; I ≤ LIN; I++) faça
4:   para (J = 1; J ≤ COL; J++) faça
5:     se (IMAGEM[I][J] > LIMITE) então
6:       IMAGEM[I][J] ← 255
7:     senão
8:       IMAGEM[I][J] ← 0
9:     fimse
10:  fimpara
11: fimpara
12: devolva IMAGEM
```

Alguns exemplos de uma imagem limiarizada em diversos níveis

Figura 11.30: Uma figura binária em 256 níveis de cinza

Figura 11.31: A figura 11.30 limiarizada com k=50

Figura 11.32: A figura 11.30 limiarizada com k=115

Figura 11.33: A figura 11.30 limiarizada com k=180

11.9 Manipulações aritméticas

Considerando a imagem como sendo uma matriz numérica, (já devidamente descomprimida, se for o caso) consistindo de valores compreendidos entre 0 (preto) e 255 (branco) para o caso de imagens monocromáticas e como três dessas matrizes para as imagens coloridas, salta aos olhos a possibilidade de efetuar manipulações aritméticas sobre tais matrizes.

Um cuidado especial deve ser tomado em relação aos limites que o resultado deve obedecer. Por exemplo, ao multiplicar (a matriz de) uma imagem por 2, aparentemente a mesma deve ficar mais clara, já que as quantidades de tinta estão sendo dobradas. Entretanto, se nenhum tratamento especial for dado, nos casos em que o valor prévio do pixel era maior ou igual a 128, haverá um estouro do campo, com resultados a princípio imprevisíveis, mas certamente indesejados. O que deve ocorrer provavelmente é que seja

 ©88-09, Pedro Kantek

deixado no campo a quantidade de tinta que excedeu a 255, podendo nesse caso o pixel ser escurecido e não clareado.

> Imagine um pixel contendo o nível 150 de tinta. Ao multiplicar por 2, ele deveria ficar com 300. Entretanto como o limite do campo é 255, será colocado lá o excesso a 255, no caso 45. Assim o pixel era 150 (cinza claro) e passou a ser 45 (quase preto). Na verdade, tais multiplicações devem sempre ser limitadas superiormente pelo valor 255.

Na direção oposta, o mesmo ocorre quando, por exemplo, se subtrai uma certa quantidade fixa de tinta de todos os pixels. Se não houver a preocupação em limitar inferiormente o valor em zero (preto), poderá ocorrer a geração de números negativos, que a depender do ambiente operacional de trabalho serão representados de maneira imprevisível, provavelmente como o complemento para uma certa quantidade.

Identicamente há que se ter a preocupação de trabalhar apenas com inteiros, já que as matrizes são de números inteiros.

Mostradas as dificuldades, eis o momento de listar algumas possibilidades.

11.9.1 Negativização

Esta função, substitui um determinado valor de pixel pelo seu complemento para 255. O que era preto vira branco e vice-versa. Eis o algoritmo:

1: função NEGATIVIZAÇÃO (inteiro IMAGEM [LIN][COL])
2: inteiro I, J
3: **para** $(I = 1;\ I \leq LIN;\ I++)$ **faça**
4: **para** $(J = 1;\ J \leq COL;\ J++)$ **faça**
5: $IMAGEM[I][J] \leftarrow 255 - IMAGEM[I][J]$
6: **fimpara**
7: **fimpara**
8: devolve $IMAGEM$

Um exemplo de imagem negativizada.

Figura 11.34: Figura original em 255 níveis de cinza

Figura 11.35: Figura 11.34 negativizada

11.9.2 Multiplicação por constante k

Esta função aumenta a quantidade de tinta de cada pixel por um fator k, com a limitação do resultado a 255. Aproveitando o mesmo algoritmo para escurecer a imagem (casos de $K \leq 1$), deve-se obter apenas a parte inteira da variável de trabalho. Para isso, vai-se usar a função primitiva "piso", cujo símbolo é $\lfloor$. Eis o algoritmo:

```
1: função MULTIPLICAÇÃO (IMAGEM [LIN] [COL] de inteiros, K real)
2: inteiro I, J
3: real AUX
4: para (I = 1; I ≤ LIN; I++) faça
5:    para (J = 1; J ≤ COL; J++) faça
6:       AUX ← ⌊ IMAGEM[I][J] × K
7:       se (AUX > 255) então
8:          AUX ← 255
9:       fimse
10:      IMAGEM[I][J] ← AUX
11:   fimpara
12: fimpara
13: devolve IMAGEM
```

Veja-se nos exemplos a seguir, duas transformações de multiplicação. A primeira, após uma multiplicação pelo fator 1.5, tendo tido o cuidado de limitar o valor máximo a 255. A segunda, após multiplicação por 2, sem observar o limite máximo. Note a distorção existente.

Figura 11.36: Figura 11.34 multiplicada por 1.5 limitado a 255

Figura 11.37: Figura 11.34 multiplicada por 2, sem limitação

11.9.3 Adição e subtração de 2 imagens

Desde que as imagens tenham o mesmo tamanho, ou possam ser transformadas em um comum, e desde que se respeitem os limites de 0 e 255, nada impede que se façam diversas operações entre imagens. Acompanhe nos exemplos, lembrando que nos dois casos os limites de 0 e 255 foram respeitados:

 ©88-09, Pedro Kantek

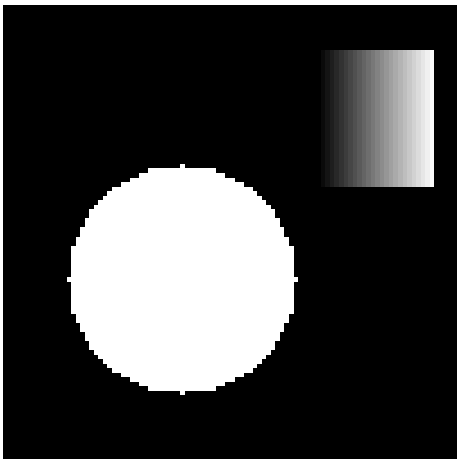


Figura 11.38: Uma figura binária qualquer

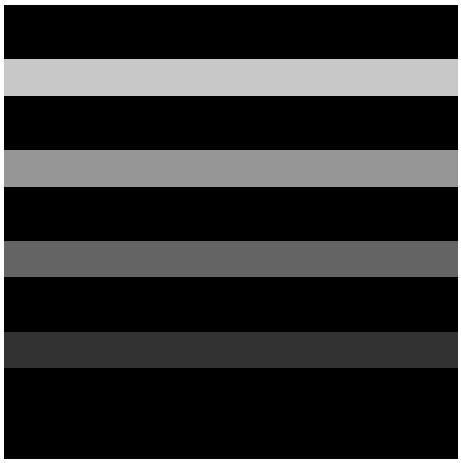


Figura 11.39: Outra figura binária

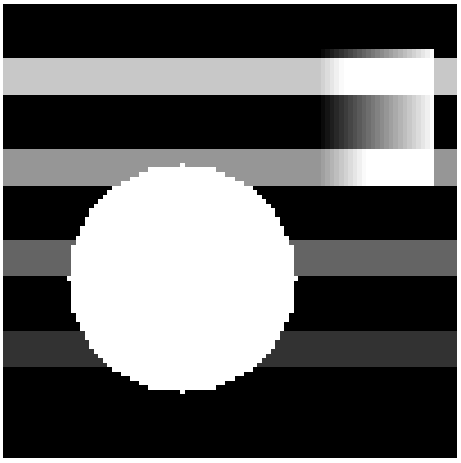


Figura 11.40: 11.38 e 11.39 somadas

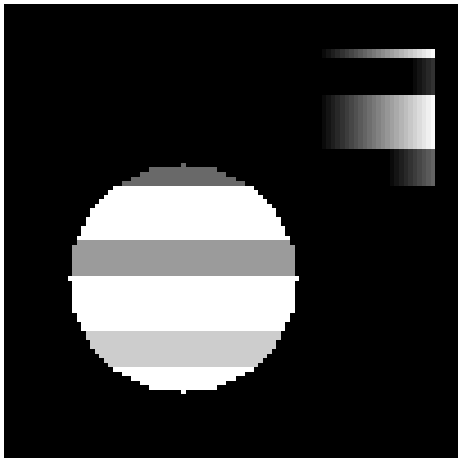


Figura 11.41: 11.38 menos 11.39

11.9.4 E e OU lógicos em imagens binárias

Valendo para imagens em apenas 2 níveis de cor, preto e branco, estas operações, e todas as demais do repertório da programação, podem ser usadas em imagens, com os resultados mais variados possíveis.

11.9.5 Zoom

Esta operação é usada para alterar o tamanho da imagem. Lembrando que a imagem é uma matriz, trata-se na verdade em alterar o tamanho desta matriz. Deve-se tratar independentemente cada um dos eixos. O caso mais simples é quando o fator de zoom é um número inteiro. Assim, ampliar uma imagem por um fator de 2 em ambas dimensões, significa que cada pixel (valor) na matriz original dará origem a 4 outros pixels (valores) na matriz ampliada. Assim, se uma imagem tem 100×100 pixels e sofre uma operação de zoom de fator 2, ela passará a ter como matriz uma de 200×200. O algoritmo que faz isso é:

```
1: função ZOOMFATOR2 (IMAGEM [LIN] [COL] de inteiros)
2: inteiro IMAZOOM [LIN × 2] [COL × 2]
3: inteiro IAUX, JAUX, I, J
4: para (I = 1; I ≤ LIN; I++) faça
5:    para (J = 1; J ≤ COL; J++) faça
6:       IAUX ← 1 + ((I − 1) × 2)
7:       JAUX ← 1 + ((J − 1) × 2)
8:       IMAZOOM[IAUX][JAUX] ← IMAGEM[I][J]
9:       IMAZOOM[IAUX + 1][JAUX] ← IMAGEM[I][J]
10:      IMAZOOM[IAUX][JAUX + 1] ← IMAGEM[I][J]
11:      IMAZOOM[IAUX + 1][JAUX + 1] ← IMAGEM[I][J]
12:   fimpara
13: fimpara
14: devolva IMAZOOM
```

11.10 Filtragem, realce e suavização

Aqui as transformações levam em consideração o valor do pixel que está sendo analisado e também o valor de seus vizinhos na imagem. Portanto, precisa-se definir o que é um *pixel vizinho* de um determinado pixel na imagem. Suponhamos uma imagem representada por sua matriz:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix}$$

Dado um pixel qualquer, $a_{x,y}$, definem-se seus 4 vizinhos como sendo os pixels $a_{x-1,y}$, $a_{x+1,y}$, $a_{x,y-1}$ e $a_{x,y+1}$. No exemplo acima, os 4-vizinhos de $a_{3,4}$ são $a_{2,4}$, $a_{4,4}$, $a_{3,3}$ e $a_{3,5}$.

De maneira similar, definem-se os 8-vizinhos de um pixel como sendo os 4 vizinhos acima definidos, mais aqueles localizados nas diagonais. Formalmente, os 8-vizinhos de um pixel $a_{x,y}$ como sendo os pixels $a_{x-1,y}$, $a_{x+1,y}$, $a_{x,y-1}$, $a_{x,y+1}$, $a_{x-1,y-1}$, $a_{x+1,y+1}$, $a_{x+1,y-1}$ e $a_{x-1,y+1}$. No exemplo acima, os 8-vizinhos de $a_{3,4}$ são $a_{2,4}$, $a_{4,4}$, $a_{3,3}$, $a_{3,5}$, $a_{2,3}$ são $a_{4,5}$, $a_{4,3}$ e $a_{2,5}$.

O conceito de n-vizinhança é importante na discussão de diversas transformadas, a seguir.

11.10.1 Filtro da média e da mediana

Este filtro tem a finalidade de diminuir o ruído porventura existente na imagem. Define-se ruído como a presença de pontos claros e/ou escuros na imagem. Utiliza-se o nome de ruído sal (claros) e/ou pimenta (escuros) para este tipo de anomalia. O filtro da mediana coloca os 8 vizinhos de um pixel em um vetor, ordena-o e obtém a mediana destes elementos. Se a diferença entre o valor atual do pixel e a mediana assim obtida for superior a um determinado valor k (parâmetro do filtro), a mediana substitui o pixel. Caso contrário, o mesmo é deixado intocado. Veja-se o algoritmo:

```
1: função FILTROMEDIANA (IMAGEM [LIN][COL] de inteiros, K inteiro)
2: inteiro I, J
3: NOVAIMA[LIN][COL] de inteiros
4: VET[8] de inteiros
```

 ©88-09, Pedro Kantek

```
5: para (I = 2; I < LIN; I++) faça
6:   para (J = 2; J < COL; J++) faça
7:     VET[1] ← IMAGEM[LIN − 1][COL − 1]
8:     VET[2] ← IMAGEM[LIN − 1][COL]
9:     VET[3] ← IMAGEM[LIN − 1][COL + 1]
10:    VET[4] ← IMAGEM[LIN][COL − 1]
11:    VET[5] ← IMAGEM[LIN][COL + 1]
12:    VET[6] ← IMAGEM[LIN + 1][COL − 1]
13:    VET[7] ← IMAGEM[LIN + 1][COL]
14:    VET[8] ← IMAGEM[LIN + 1][COL + 1]
15:    VET ← ordenaVET
16:    se (abs(IMAGEM[LIN][COL] − VET[4]) > K) então
17:      NOVAIMA[LIN][COL] ← VET[4]
18:    senão
19:      NOVAIMA[LIN][COL] ← IMAGEM[LIN][COL]
20:    fimse
21:  fimpara
22: fimpara
23: devolva NOVAIMA
```

Veja a seguir, como ficam uma imagem devidamente mesclada a ruido sal com pimenta com a aplicação do filtro da mediana.

Figura 11.42: Figura original em 256 níveis de cinza

Figura 11.43: A figura 11.42 com ruido sal e pimenta

Figura 11.44: A figura 11.43 filtrada via mediana com K=0

Figura 11.45: A figura 11.43 filtrada via mediana com K=70

11.10.2 Espalhamento de tinta

Este algoritmo visa "espalhar"uma determinada cor a partir de um pixel, aos vizinhos deste pixel e que estejam originalmente conectados a menos de um determinado limite. Em outras palavras a cor é espalhada, desde que a cor original do pixel considerado seja igual (ou menor que um dado limite) à de seus vizinhos.

Veja-se, por exemplo, a seguinte imagem imaginária:

5	6	7	4	4	5
8	8	7	4	4	5
6	6	7	4	5	5
6	5	5	4	5	5
6	5	5	4	5	6

Aplicando o algoritmo de espalhamento de tinta com $limite = 2$, fica

5	5	7	4	4	4
7	7	7	4	4	4
7	7	7	4	4	4
7	4	4	4	4	4
7	4	4	4	4	6

Note-se que a aplicação acima (limite=2), teve 2 passadas.

Agora, aplicando o mesmo algoritmo com limite=3, fica

5	5	5	5	5	5
8	8	8	5	5	5
8	8	8	5	5	5
8	5	5	5	5	5
8	5	5	5	5	5

Aqui houve 8 passadas.

Seja agora o resultado com limite=4

5	5	5	5	5	5
5	5	5	5	5	5
5	5	5	5	5	5
5	5	5	5	5	5
5	5	5	5	5	5

Aqui houve 3 passadas. Note que agora toda a tinta foi espalhada.

Eis o algoritmo de espalhamento de tinta:

 ©88-09, Pedro Kantek

1: função de espalhamento de tinta
2: de cima para baixo e da esquerda para a direita:
3: se a menor diferença entre o pixel P[i][j] e os vizinhos P[i-1][j-1], P[i-1][j], P[i-1][j+1] e P[i][j-1] for menor que o limite, mas diferente de zero,
4: então P[i][j] ← vizinho de menor diferença
5: de baixo para cima e da direita para a esquerda:
6: se a menor diferença entre o pixel P[i][j] e os vizinhos P[i+1][j-1], P[i+1][j], P[i+1][j+1] e P[i][j+1] for menor que o limite, mas diferente de zero,
7: então P[i][j] ← vizinho de menor diferença
8: Refaça os 2 ciclos acima, até não haver mudança na imagem

11.10.3 Componentes Conectados

O objetivo desta transformação é a identificação dos diversos componentes conectados conexos. Para tanto, a entrada deve ser uma imagem contendo um fundo e demais componentes. Tudo aquilo que for diferente do fundo, será considerado como algum componente. A resposta será dada em uma matriz de mesma dimensão da image. Os pixels correspondentes ao fundo terão zero, enquanto os pixels referentes aos componentes serão numerados a partir de 1 para cada componente. Para a imagem mostrada

Figura 11.46: Uma imagem binária

em 11.46, o algoritmo de componentes conectados devolveu a seguinte matriz numérica (devidamente invertida):

```
0  0 0 0 0  0  0  0 0 0 0 0 0 0 0 0 0 0 0 0
0  0 0 0 0  0  0  0 0 0 0 0 0 0 0 0 0 0 0 0
0  0 1 1 0  0  0  0 0 0 3 3 0 0 0 0 0 0 0 0
0  0 1 1 0  0  0  0 0 0 3 3 0 0 0 0 0 0 0 0
0  0 1 1 0  0  0  0 0 0 3 3 3 3 3 3 3 0 0 0
0  0 1 1 0  0  0  0 0 0 3 3 3 3 3 3 3 0 0 0
0  0 1 1 1  1  1  0 0 0 3 3 0 0 0 0 0 0 0 0
0  0 1 1 1  1  1  0 0 0 3 3 3 0 0 0 0 0 0 0
0  0 0 0 0  0  0  0 0 0 3 3 3 0 0 0 0 0 0 0
0  0 0 0 0  0  0  0 0 0 3 3 3 0 0 0 0 0 0 0
0  0 0 0 0  0  0  0 0 0 3 3 3 3 3 3 3 0 0 0
0  0 0 0 0  0  0  0 0 0 3 3 3 3 3 3 3 0 0 0
0  0 0 0 0  0  0  0 0 0 3 3 0 0 0 0 0 0 0 0
```

```
0  0 0 0 0 64 64 64 0 0 3 3 0 0 0 0 0 0 0 0
0  0 0 0 0 64 64 64 0 0 3 3 3 0 0 0 0 0 0 0
0  0 0 0 0 64 64 64 0 0 3 3 3 3 3 3 3 3 3 0
0  0 0 0 0  0  0  0 0 0 3 3 3 3 3 3 3 3 3 0
0  0 0 0 0  0  0  0 0 0 0 0 0 0 0 0 0 0 0 0
0 96 0 0 0  0  0  0 0 0 0 0 0 0 0 0 0 0 0 0
0  0 0 0 0  0  0  0 0 0 0 0 0 0 0 0 0 0 0 0
```

Eis o algoritmo:

```
 1: função COMPCONEC (IMAGEM [LIN][COL] de inteiros)
 2: inteiro I, J, K, ACUM, MEN
 3: lógico MUDOU ← VERDADEIRO
 4: CC[LIN][COL] de inteiros
 5: CC ← 0
 6: ACUM ← 1
 7: para (I = 2; I < LIN; I++) faça
 8:   para (J = 2; J < COL; J++) faça
 9:     se (IMAGEM[I][J] ≠ FUNDO) então
10:       CC[I][J] ← ACUM
11:       ACUM++
12:     fimse
13:   fimpara
14: fimpara
15: enquanto MUDOU faça
16:   MUDOU ← FALSO
17:   para (I = 2; I < LIN; I++) faça
18:     para (J = 2; J < COL; J++) faça
19:       se (CC[I][J] ≠ 0) então
20:         MEN ← CC[I][J]
21:         se (CC[I − 1][J − 1] ≠ 0) ∧ (CC[I − 1][J − 1] < MEN) então
22:           MEN ← CC[I − 1][J − 1]
23:         fimse
24:         se (CC[I − 1][J] ≠ 0) ∧ (CC[I − 1][J] < MEN) então
25:           MEN ← CC[I − 1][J]
26:         fimse
27:         se (CC[I − 1][J + 1] ≠ 0) ∧ (CC[I − 1][J + 1] < MEN) então
28:           MEN ← CC[I − 1][J + 1]
29:         fimse
30:         se (CC[I][J − 1] ≠ 0) ∧ (CC[I][J − 1] < MEN) então
31:           MEN ← CC[I][J − 1]
32:         fimse
33:         se (CC[I][J] ≠ MEN) então
34:           CC[I][J] ← MEN
35:           MUDOU ← VERDADEIRO
36:         fimse
37:       fimse
38:     fimpara
39:   fimpara
40:   para (I = LIN − 1; I < 2; I−−) faça
41:     para (J = COL − 1; J < 2; J−−) faça
42:       se (CC[I][J] ≠ 0) então
43:         MEN ← CC[I][J]
44:         se (CC[I + 1][J + 1] ≠ 0) ∧ (CC[I + 1][J + 1] < MEN) então
```

 ©88-09, Pedro Kantek

```
45:              MEN ← CC[I + 1][J + 1]
46:            fimse
47:            se (CC[I + 1][J] ≠ 0) ∧ (CC[I + 1][J] < MEN) então
48:              MEN ← CC[I + 1][J]
49:            fimse
50:            se (CC[I + 1][J − 1] ≠ 0) ∧ (CC[I + 1][J − 1] < MEN) então
51:              MEN ← CC[I + 1][J − 1]
52:            fimse
53:            se (CC[I][J − 1] ≠ 0) ∧ (CC[I][J − 1] < MEN) então
54:              MEN ← CC[I][J − 1]
55:            fimse
56:            se (CC[I][J] ≠ MEN) então
57:              CC[I][J] ← MEN
58:              MUDOU ← VERDADEIRO
59:            fimse
60:          fimse
61:        fimpara
62:      fimpara
63: fimenquanto
64: devolva CC
```

11.10.4 Trabalhos de anos passados

Para auxílio, eis o trabalho que foi pedido às turmas de IA em 2003:

O café é um dos principais produtos de exportação de nosso país. Somos o maior exportador do produto com cerca de 25 milhões de sacas/ano. Somos o segundo maior consumidor, atrás apenas dos EUA. Historicamente, o café é analisado segundo testes manuais. A ABIC utiliza um teste padrão. Ele começa pela preparação de 4 amostras (1 fica lacrada no produtor, 2 ficam lacradas na ABIC e são a contra-prova e a quarta é analisada). O teste começa pela lavagem da amostra em clorofórmio para desagregação e desengraxamento. A seguir ela é examinada (por um humano) com aumento de 12x. Havendo suspeita de fraude, testes de extratos aquosos confirmam ou não a fraude. (5 gramas de pó + 300 ml de água destilada são fervidos por 1 hora). Depois disso o material é coado. No máximo 30% do volume deve ficar retido. Se for mais de 30% isso caracteriza a fraude. Vide mais detalhes em Ciência Hoje, vol 31, número 186).

O trabalho que vai ser aqui pedido não é idêntico ao desenvolvido pela ABIC, sendo apenas baseado nele. O programa que você deve escrever deverá certificar a qualidade do café sob análise, baseado apenas nas dimensões e freqüência dos grãos.

O programa receberá uma série de 5 fotos de amostras de grãos de café, devidamente desagregados e desengraxados. Por isso, as amostras deverão ter os grãos soltos (pelo menos 1 pixel de fundo entre cada grão). Os valores de tamanho aqui descritos já levam em consideração a distância focal das fotos que será constante. As imagens serão monocromáticas com 256 níveis de cinza, no formato BMP, com dimensões de 28 linhas por 74 colunas. Note-se que existe uma 29. linha (na verdade a primeira) que contém a identificação da imagem, do bloco e da amostra. Deve-se notar que a informação referente à amostra é apenas para identificação externa. Para nós aqui, interessam o bloco e a imagem. Os nomes dos arquivos serão A03BxIy, onde x é o bloco e y é a imagem. A identificação será sempre padrão, composta pela mensagem "SCAC-Amostrakkkk-Blocoxxx-Imagemyyy", onde kkkk é a amostra, xxx é o bloco e yyy é a imagem.

Tratamento que o programa deverá fazer:

1. Tratamento das imagens para efeito de processamento. No mínimo, há que se

fazer uma limiarização, seguida de uma contagem e individualização de cada grão dentro da imagem.

2. verificar se existem pelo menos 10 grãos na foto. Se não existirem rejeitar esta imagem.

3. Verificar se 1 grão é maior do que 90 pixels. Se isso for verdade é sinal de que os grãos não estão separados, e conseqüentemente a imagem deve ser rejeitada.

4. Verificar se pelo menos 3 imagens do bloco são aceitáveis. Se não forem, o bloco todo deve ser rejeitado.

5. Se o bloco for aceito, para cada imagem os grãos devem ser individualizados, atribuindo a cada um tamanho em pixels e suas duas dimensões máximas (vertical e horizontal, por hipótese). Não há necessidade de rotacionar as fotos.

6. grãos com coeficiente de comprimento ≥ 3 devem ser rejeitados como sendo detrito. O coeficiente de comprimento de um grão é a divisão da maior dimensão pela menor dimensão.

7. grãos com tamanho menor do que 30 pixels são rejeitados como grão quebrado.

8. grãos com tamanho maior do que 60 pixels são rejeitados como impurezas. Se um grão cair nesta categoria e também na categoria detrito, será detrito.

9. Os demais grãos restantes são considerados café.

10. Uma média deve ser obtida considerando todas as imagens obtidas (que serão 3, 4 ou 5). A média deve ser ponderada pelo número de pixels de cada tipo.

11. Cada entidade na foto deve ser nomeada (café inteiro, café quebrado, impureza, detrito). Depois as fotos e suas entidades devem ser agrupadas a fim de obter as percentagens (ponderadas por pixel) de café inteiro, de café quebrado, de impureza e de detritos.

A seguir, o café deve ser classificado, segundo a tabela a seguir:

Tipo de café	característica
Premium	100% de café
Primeira	80% ou mais de café E 20% ou menos de grãos quebrados E 0 % de impurezas E 0% de detritos
Segunda	60% ou mais de café E 30% ou mais de de grãos quebrados E 10% ou menos de impurezas + detritos
Terceira	qualquer combinação exceto a categoria rejeitado
Rejeitado	Impurezas igual ou maior do que 20% OU detritos igual ou maior do que 15%

O resultado deve ser: SCAC - Amostra xxxx, composta de x imagens, é de qualidade XXXXXXXXXX.

No primeiro dia de laboratório, o professor entregará 3 amostras devidamente classificadas, a fim de que vocês possam ter uma massa de teste certificada. No dia da avaliação (após o encerramento do prazo de construção), outras 3 amostras inéditas serão testadas e o programa avaliado em função do seu comportamento frente a estas amostras inéditas.

Propostas de adicionais:

 ©88-09, Pedro Kantek

1. Mostrar as imagens das amostras, eventualmente colorindo cada tipo de grão com uma legenda qualquer
2. Apresentar um laudo de cada foto, ainda antes de fazer as médias do bloco
3. Apresentar um laudo global descrevendo os diversos valores que levam à classificação dada.

Outro trabalho (3.bim de 2002):

O Trabalho "Olho no céu"A administração do Aeroporto Internacional de Curitiba, está preocupada com a ocorrência de balões de São João nas cercanias do aeroporto. Deve-se notar que esses artefatos não aparecem nos radares de aproximação de vôo, e com isso podem por em risco a aproximação e afastamente de aviões.

Definiu-se um círculo de 2,5 Km, cujo centro é uma sala envidraçada com 360 graus de visão livre. Nesta sala foi instalado um sistema de captura de imagens, que consegue gerar sequencias de imagens de setores do céu. Cada conjunto é composto por 3 imagens, obtidas do mesmo setor, com intervalo de tempo de 8 segundos entre cada imagem. O objetivo destas 3 imagens é caracterizar a mobilidade dos objetos capturados.

As imagens serão monocromáticas, com 256 níveis de cinza e definiu-se uma resolução tal que balões ameaçadores (mais de $4m^3$ de volume), surjam na imagem com um tamanho mínimo de 12 pixels contíguos e homogêneos, de nivel de cinza menor para visões diurnas e de 12 pixels contíguos e não homogêneos, de nivel de cinza maior, para visões noturnas.

As condições metereológicas influenciam a captura de imagens, assim, devem originar um processamento devido nas imagens antes delas sofrerem análise. Para tanto, considerar as seguintes regras:

a) A transformação dia-noite se dará as 17h30 e a noite-dia as 06h00 diariamente.

b) Condições de visibilidade inferiores a 2000 m, inviabilizam os resultados obtidos pelo sistema. Não há necessidade de preocupar-se com isto, pois não haverá geração de imagens nestes casos.

As condições técnicas da captura também influenciam as imagens. Embora todos os cuidados técnicos tenham sido observados, é esperado um ruído aleatório, do tipo sal e pimenta (sal=nivel de cinza > 240, e pimenta=nivel de cinza < 15) com freqüência máxima de 0,1% por pixel.

Finalmente, segundo a Aeronáutica, devem ser localizados e tratados os seguintes possíveis objetos:

a) aeronaves: tamanho variável, mas superior a 25 pixels, Identificadas por uma velocidade mínima de 4 pixels/quadro em uma direção contida no plano da captura. Este sistema deve ignorar tais objetos, pois os mesmos são controlados pelo radar de aproximação.

b) urubus: tamanho máximo de 5 pixels e com velocidade maior ou igual a 0 pixel/quadro em qualquer direção contida no plano da captura. O sistema deve ignorar urubus.

c) Balões de São João: tamanho mínimo de 12 pixels e velocidade igual a zero pixels/quadro em intervalos menores do que 30 segundos. O sistema deve informar a posição no céu (altura astronômica e azimute) da localização do objeto.

d) OVNIS: tamanho e velocidade desconhecidas. O sistema deve ignorar estes objetos.

e) Núvens: Existem dois tipos de nuvens, as próximas e as distantes. As distantes, tem bordas identificáveis, tem tamanho maior que 50 pixels e são imóveis. As próximas, podem ter qualquer tamanho, e alguma mobilidade, mas suas bordas não são definidas. A regra de não definição é que dois pixels vizinhos e que pertençam à nuvem não podem ter variação maior do que 5 níveis de tinta. Da mesma maneira, as bordas dessa nuvem também serão difusas, com variação inferior a 5 niveis de tinta, entre a nuvem e o céu.

Perceba que aqui temos uma dificuldade. Um objeto maior que 50 pixels é uma nuvem ou um balão ???

Deve-se notar que estes tamanhos foram obtidos para captura próxima ao perímetro do circulo acima descrito. Obviamente, objetos dentro do círculo parecerão maiores, mas já deverão ter sido identificados e acompanhados no momento da entrada no circulo. Note-se que neste momento, eles terão os tamanhos acima descritos.

Cada imagem a ser tratada será gravada usando o formato BMP (bit mapped image) com 8 bits/pixel. Serão monocromáticas com 256 níveis de cinza. Cada imagem terá 81 linhas por 120 colunas e representará a projeção de um retângulo sobre a abóboda celeste com 1 grau de largura por 40 minutos de altura. A primeira (última no desenho) das linhas não terá imagem e sim os dados de controle desta imagem.

Dentro do arquivo BMP, haverá as seguintes informações:

a) Data da captura, no formato AAMMDD

b) Hora da captura no formato HHMMSS, onde $0 \leq$ HH ≤ 23 e $0 \leq$ MM,SS ≤ 59

c) Altura astronômica do ponto mais alto e à esquerda da imagem, no formato GGGMMSS

d) Azimute (em relação ao Sul) do ponto mais alto e à esquerda da imagem no formato GGGMMSS

Estas informações estarão nos seguintes endereços do arquivo: (correspondentes à linha 1) Assinatura do arquivo (32 bytes) de X'436' a X'455', igual a constante DEOLHONOURUBU, seguido de zeros binários

Altura astronomica: X'456' e X'457' = graus, na forma de um número binário de 2 bytes (sem inversão) X'458' = minutos, em binário X'459' = segundos em binário

Azimute: X'45A' e X'45B' = graus na forma de um número binário de 2 bytes (sem inversão) X'45C' = minutos, em binário X'45D' = segundos, em binário

Horário: X'43E' = hora em binário X'45F' = minutos de hora, em binário X'460' = segundos de hora em binário

Data: X'461' = ano, variando entre 00 e 99, binário X'462' = mes, variando entre 1 e 12, binário X'463' = dia, variando entre 1 e 31, binário.

Reservado, de X'464' até X'4AD' para uso de controle da aplicação, conteúdo imprevisível. A primeira linha real do desenho começa em X'4AE'. Uma possivel assinatura seria:

```
436
---
 FD FD FD 00 FE FE FE 00  FF FF FF 00 44 45 4F 4C  48 4F 4E 4F 55 52 55 42  55 00 00 00 00 00 00
 00 00 00 00 00 00 00 00  00 00 00 00 00 1B 0C 00  00 AC 00 00 0B 08 02 08  16 FF FF FF FF FF FF
 FF FF FF FF FF FF FF FF  FF FF F9 DE B1 74 16 0E  10 0B 10 16 24 FF FF FF  FF FF FF FF FF FF FF
 FF FF FF FF FF FF FF FF  FF FF 20 20 20 20 20 20  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20
```

O programa a ser construído, receberá três conjuntos de 3 imagens e deverá acusar a presença ou ausência de balões (eventualmente mais de um) de maneira correta em pelo menos dois deles. Caso o programa detecte-os deve dizer sua localização no céu.

Opcionais: 1. Visualização das 3 imagens, eventualmente em movimento.

2. Identificação nas imagens visualizadas dos balões detectados

3. Identificação nas imagens visualizadas de mais objetos porventura existentes, eventualmente colorindo-os com cores artificiais.

4. Apresentação de dados técnicos referentes a cada imagem, antes e depois de cada processamento, por exemplo (entre outros): Amplitude, Dimensões, Histograma, Fator ideal de corte, etc

5. Zoom In (aumento) de imagens selecionadas

6. Outros, a critério dos alunos.

 ©88-09, Pedro Kantek

Serão disponibilizado aos alunos, no primeiro dia de prática, três conjuntos de três imagens cada, contendo balões, aviões e urubus (mas não UFOS). Cada imagem será discutida até ser aceita pelos alunos como aderentes a esta especificação. As imagens tem nomes: C1IMAGE1.BMP, C1IMAGE2.BMP, C3IMAGE3.BMP (conjunto 1), C2... (conjunto 2) e C3... (conjunto3).

No dia da examinação, outras 3 imagens distintas serão usadas para avaliar a acurácia dos programas construídos.

Proposta de solução Examinar as 3 imagens individualmente e:

1. Eliminação de ruído: Sugestão: aplicação dos algoritmos de supressão de ruido via média e/ou mediana.

2. Eliminação das nuvens próximas: Sugestao: aplicar o algoritmo de espalhamento de tinta.

3. Melhorar a imagem (contraste). Pode ser com o algoritmo de equalização de histograma ou pela aplicação de uma distribuição linear de cinza, de maneira a destacar bem os objetos.

4. Aplicar o negativo, se for o caso (imagens noturnas)

5. Aplicar limiarização. O cálculo do limite de limiarização fica em aberto. Sugestões ?

6. Descobrir os componentes da imagem. Pode-se usar o algoritmo de componentes conectados.

7. Analisar as 3 imagens para detectar movimento

8. Eventualmente mostrar a imagens aumentadas, usando o algoritmo de zoom-in.

9. Concluir, se possível.

11.11 Operadores Morfológicos

11.11.1 Erosão Binária

Seja uma imagem binária I formada por uma grade retangular de $m \times n$ pixels. Seja um elemento estruturante E também retangular formado por $k \times l$ pixels. A *erosão* da imagem I pelo elemento E é uma nova imagem obtida deslizando-se o elemento estruturante pela imagem original, e tomando apenas os pixels da imagem original que quando colocados no centro do elemento estruturante garantiram a inclusão de todos os pixels significativos do elemento estruturante dentro da imagem original.

Seja o exemplo: Supondo a imagem binária, onde o zero está representado por um "."e o um está representado por um "X".

$$I = \begin{bmatrix} . & . & . & . & . & . \\ . & X & X & X & . & . \\ . & X & X & X & X & . \\ . & . & X & X & X & . \\ . & . & . & X & X & . \\ . & . & . & X & X & . \\ . & . & . & . & . & . \end{bmatrix}$$

E seja o elemento estruturante 3×3 assim formado

$$E = \begin{bmatrix} . & X & . \\ . & (X) & . \\ . & X & . \end{bmatrix}$$

O elemento central da máscara, reconhecido por um "()" é o elemento principal.

O elemento estruturante deve percorrer todos os pontos da imagem original, de maneira a que o elemento principal do mesmo (usualmente o elemento que está no

centro) coincida com todos os elementos da imagem original.

Em cada um desses momentos, (o elemento central da máscara estruturante posicionado sobre um determinado pixel da imagem original), deve-se fazer a seguinte pergunta: Existe ALGUM pixel 1 da máscara estruturante sobre um pixel valendo 0 da imagem ? Se a resposta for SIM, esta posição correspondente na saída (a imagem erodida) deve ser 0, senão ela será 1.

Por exemplo, no caso acima, quando o elemento principal (o $2,2$ da máscara) está sobre a posição $1,1$ da imagem, existem pelo menos um (na verdade, 3) pixels 1 da máscara sobre pontos que não são 1 na imagem. Por isso, a posição $1,1$ da imagem erodida será zero. Seguindo adiante, a primeira posição valendo 1 na saída corresponde ao pixel $3,3$ da entrada.

A imagem erodida, fica

$$E \textbf{ ero } I = \begin{bmatrix} . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & X & X & . & . \\ . & . & . & X & X & . \\ . & . & . & X & X & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \end{bmatrix}$$

Os efeitos da operação de erosão podem ser assim descritos

- Diminui as partículas da imagem
- Elimina grãos de tamanho menor do que o tamanho do elemento estruturante
- Aumentar os buracos da imagem
- Permite a separação de grãos próximos.

Elemento estruturante transposto $\tilde{E}$

Antes de prosseguir, precisamos definir o elemento estruturante $\tilde{E}$ a partir do elemento E. Para obter $\tilde{E}$, rota-se E na vertical e depois na horizontal. Veja no exemplo

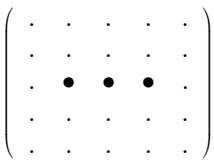
$$Se\ E = \begin{Bmatrix} . & \bullet & . \\ . & . & \bullet \\ . & . & . \end{Bmatrix}\ temos\ \ \tilde{E} = \begin{Bmatrix} . & . & . \\ \bullet & . & . \\ . & \bullet & . \end{Bmatrix}$$

Uma segunda definição de erosão Usando os operadores de Minkowski, a erosão binária de uma imagem I com um elemento estruturante E, pode ser representada usando a subtração de Minkowski ($\ominus$) como E **ero** I $= I \ominus \tilde{E} = \cap_{e \in \tilde{E}} I_e$
Note-se que nesta definição quem se desloca é a imagem E e não o elemento estruturante I, o que acaba gerando economia substancial de tempo de processamento. Acompanhe no exemplo a utilização da subtração de Minkowski ($\ominus$).

$$\begin{pmatrix} . & . & . & . & . \\ . & \bullet & \bullet & \bullet & . \\ . & \bullet & \bullet & \bullet & . \\ . & \bullet & \bullet & \bullet & . \\ . & . & . & . & . \end{pmatrix} \textbf{ero} \begin{Bmatrix} . & \bullet & . \\ . & \bullet & . \\ . & \bullet & . \end{Bmatrix} =$$

$$\begin{pmatrix} . & . & . & . & . \\ . & \bullet & \bullet & \bullet & . \\ . & \bullet & \bullet & \bullet & . \\ . & \bullet & \bullet & \bullet & . \\ . & . & . & . & . \end{pmatrix} \cap \begin{pmatrix} . & \bullet & \bullet & \bullet & . \\ . & \bullet & \bullet & \bullet & . \\ . & \bullet & \bullet & \bullet & . \\ . & . & . & . & . \\ . & . & . & . & . \end{pmatrix} \cap \begin{pmatrix} . & . & . & . & . \\ . & . & . & . & . \\ . & \bullet & \bullet & \bullet & . \\ . & \bullet & \bullet & \bullet & . \\ . & \bullet & \bullet & \bullet & . \end{pmatrix} =$$

 ©88-09, Pedro Kantek



Acompanhe aqui um exemplo de aplicação de erosão a uma imagem.

Figura 11.47: Figura original em 255 níveis de cinza

Figura 11.48: Figura ja limiarizada (L=100)

Figura 11.49: Uma erosão em cruz

Figura 11.50: Duas erosões em cruz

Deve-se atentar para o fato de que ao executar a erosão é necessária a colocação de linhas e colunas adicionais contendo o fundo (que pode ser 0 ou 1, a depender da figura).

Figura 11.51: Três erosões em cruz

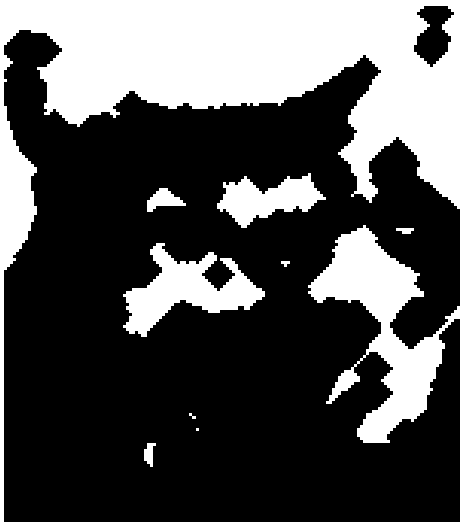


Figura 11.52: Cinco erosões em cruz

11.11.2 Dilatação Binária

A regra aqui é também variar o elemento estruturante (máscara) sobre a imagem original. Uma vez alinhados o item principal da máscara com o pixel sob exame, a regra é: Se existir algum pixel 1 da mascara estruturante sobre um 1 da imagem, o pixel do resultado será 1. Caso contrário será zero. Acompanhe no exemplo duas aplicações da dilatação sobre a ultima imagem do gato. Os efeitos da operação de dilatação podem ser assim

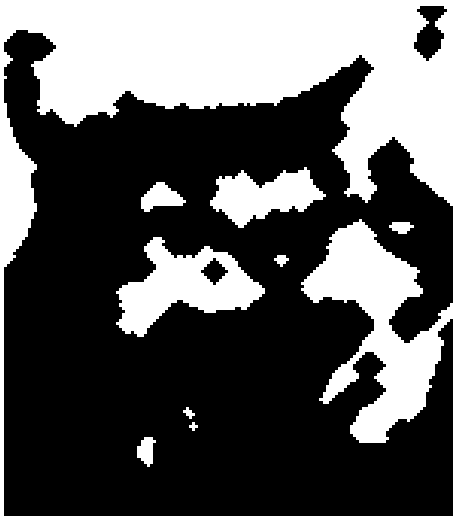


Figura 11.53: Ultima figura que sofreu uma dilatação em cruz

Figura 11.54: Mais uma dilatação em cruz

descritos

- Aumenta as partículas da imagem
- Preenche pequenos buracos

 ©88-09, Pedro Kantek

- Conecta partes pequenas próximas

Uma segunda definição de dilatação Usando os operadores de Minkowski, a dilatação binária de uma imagem I com um elemento estruturante E, pode ser representada usando a adição de Minkowski ($\oplus$) como E **dil** I $= I \oplus \tilde{E} = \cup_{e \in \tilde{E}} I_{-e}$

Acompanhe no exemplo a utilização da adição de Minkowski ($\oplus$).

$$\begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \bullet & \cdot & \cdot \\ \cdot & \bullet & \bullet & \bullet & \cdot \\ \cdot & \bullet & \cdot & \bullet & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \textbf{dil} \left\{ \begin{matrix} \cdot \\ \bullet \\ \bullet \end{matrix} \right\} =$$

$$\begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \bullet & \cdot & \cdot \\ \cdot & \bullet & \bullet & \bullet & \cdot \\ \cdot & \bullet & \cdot & \bullet & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \oplus \left\{ \begin{matrix} \tilde{} \\ \cdot \\ \bullet \\ \bullet \end{matrix} \right\} =$$

$$\begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \bullet & \cdot & \cdot \\ \cdot & \bullet & \bullet & \bullet & \cdot \\ \cdot & \bullet & \cdot & \bullet & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \oplus \left\{ \begin{matrix} \cdot \\ \bullet \\ \bullet \end{matrix} \right\} =$$

$$\begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \bullet & \cdot & \cdot \\ \cdot & \bullet & \bullet & \bullet & \cdot \\ \cdot & \bullet & \cdot & \bullet & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \cup \begin{pmatrix} \cdot & \cdot & \bullet & \cdot & \cdot \\ \cdot & \bullet & \bullet & \bullet & \cdot \\ \cdot & \bullet & \cdot & \bullet & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} =$$

$$\begin{pmatrix} \cdot & \cdot & \bullet & \cdot & \cdot \\ \cdot & \bullet & \bullet & \bullet & \cdot \\ \cdot & \bullet & \bullet & \bullet & \cdot \\ \cdot & \bullet & \cdot & \bullet & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

11.11.3 Detecção de Bordas

Uma operação importante ao se manusear imagens digitais é a obtenção da borda de uma figura. Usando o conceito de erosão visto acima, salta aos olhos que a borda de uma imagem é ela mesma tirando-se os pontos que formam a erosão dessa imagem original.

Em outras palavras, considerando a imagem I, a borda de I (denotada **bor**(I)) é: **bor**$(I) = I - E$ **ero** I, onde o sinal de menos indica a subtração pixel a pixel da imagem.

A máscara indicada para a operação de erosão para detecção de bordas é $E = \begin{bmatrix} . & X & . \\ X & (X) & X \\ . & X & . \end{bmatrix}$ ou então, $E = \begin{bmatrix} X & X & X \\ X & (X) & X \\ X & X & X \end{bmatrix}$ sendo que esta última é mais eficiente (pois pode ser decomposta), mas deixa pixels nos cantos da borda que eventualmente poderiam (ou deveriam) ser extraídos para a geração de uma borda 100% correta.

Veja-se no exemplo

Figura 11.55: subtração de 11.48 menos 11.49

Figura 11.56: subtração de 11.48 menos 11.51

11.11.4 Abertura Binária

Esta operação visa eliminar pontos indesejáveis na imagem sem no entanto alterar o tamanho das partes que compõe a imagem. A abertura é formada por uma erosão seguida de uma dilatação da imagem. Em linguajar matemático, teremos: E **abe** $I = \tilde{E}$ **dil** $(E$ **ero** $I)$

A máscara $\tilde{E}$ é obtida fazendo-se o transposto de E por simetria central pela origem. Trocando em miúdos, se o elemento estruturante E é $E = \begin{bmatrix} . & X & . \\ . & . & X \\ . & . & . \end{bmatrix}$ O transposto $\tilde{E}$ é $\tilde{E} = \begin{bmatrix} . & . & . \\ X & . & . \\ . & X & . \end{bmatrix}$

Usando as operações de Minkowski, podemos escrever E **abe** $I = E \oplus (\tilde{E} \ominus I)$

Poder-se-ia pensar que uma erosão seguida de uma dilatação tenderia a devolver a imagem original. Isso em geral não ocorre. O resultado é mais regular, com menos detalhes (e menos ruído). Veja a seguir, um exemplo da aplicação de abertura.

Figura 11.57: figura original com 256 níveis de cinza

Figura 11.58: figura limiarizada com L=110

 ©88-09, Pedro Kantek

Figura 11.59: abertura com 0 1 0-0 1 0-0 1 0

Figura 11.60: abertura com 1 1 1-1 1 1-1 1 1

11.11.5 Fechamento Binário

Esta operação, ao contrário da abertura binária, consiste em dilatar e depois erodir o resultado da dilatação. A fórmula: E **fec** $I = \tilde{E}$ **ero** $(E$ **dil** $I)$

O fechamento suaviza as fronteiras da imagem e preenche buracos que tenham tamanho inferior à máscara. Veja-se a seguir, exemplos de fechamento.

Figura 11.61: figura original com 256 níveis de cinza

Figura 11.62: figura limiarizada com L=110

Figura 11.63: fechamento com 0 1 0-0 1 0-0 1 0

Figura 11.64: fechamento com 1 1 1-1 1 1-1 1 1

Idempotência Tanto a operação de abertura e do fechamento são idempotentes. [1]. Esta é a diferença de ambas para as operações de erosão e dilatação. As expressões que descrevem este efeito são:

$$\mathbf{fec}^B(\mathbf{fec}^B(X)) = \mathbf{fec}^B(X)$$

e

$$\mathbf{abe}^B(\mathbf{abe}^B(X)) = \mathbf{abe}^B(X)$$

.

11.11.6 Transformação hit-miss

Para realizar esta operação precisamos de dois elementos estruturantes E^i e E^e que formam um conjunto para a aplicação de hit-miss. Define-se o conjunto V como $V = (E^i, E^e)$ e neste, ambos os subconjuntos devem ser disjuntos, senão a transformação não está definida. O conjunto E^i será usado para testar a parte interna da imagem e o conjunto E^e será usado para a parte externa, ou melhor dizendo a imagem complementar à imagem interna. Com isto, temos a seguinte definição:
A transformação **him** sobre o conjunto X a partir dos elementos estruturantes $V = (E^i, E^e)$ é

$$\mathbf{him}^v(X) \;=\; X \;\mathbf{him}\; V \;= \{x : E^i_x \subset X; E^e_x \subset X^c\}$$

A partir desta definição, um ponto de X pertence a X **him** V se e somente se E^i "cabe" em X e E^e "cabe" em X^c.

Uma definição alternativa da transformação hit-miss, usando a definição da erosão pode ser:

$$E \;\mathbf{him}\; I = (E^i \;\mathbf{ero}\; I) \cap (E^e \;\mathbf{ero}\; I^c)$$

, onde I^c é a imagem complementar de I (o negativo...).

Para poder trabalhar esta operação morfológica, precisamos definir uma nova máscara formada por 3 símbolos:

o Representa os pixels que formam o conjunto E^e

X Representa os pixels que formam o conjunto E^i

? Representa os pixels irrelevantes, que não fazem parte de nenhum dos conjuntos E^e ou E^i.

Seja por exemplo a imagem I

$$\begin{pmatrix}
. & . & . & . & . & . \\
. & . & . & . & . & . \\
. & \bullet & \bullet & . & . & . \\
. & \bullet & \bullet & . & . & . \\
. & \bullet & \bullet & . & . & . \\
. & . & . & . & . & . \\
. & \bullet & \bullet & \bullet & \bullet & . \\
. & \bullet & \bullet & \bullet & . & . \\
. & \bullet & . & . & . & . \\
. & . & . & \bullet & . & . \\
. & . & . & . & . & .
\end{pmatrix}$$

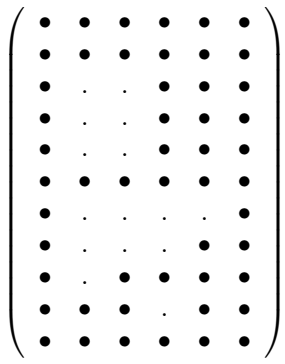
[1] Uma operação . é idempotente, se para $\forall x$, é válida a expressão $x.x = x$

 ©88-09, Pedro Kantek

Fazendo a transformação **hit-miss** desta imagem com o conjunto V,

$$V = \left\{ \begin{array}{ccc} ? & o & o \\ ? & ? & ? \\ X & X & ? \end{array} \right\}$$

Considerando a imagem complementar I^c,



Fazendo as duas erosões, fica ...

11.11.7 Afinamento

Uma característica importantes das transformações é o **homotopismo**. Uma transformação é homotópica quando ela não altera a quantidade de componentes de uma imagem. (Por componente, aqui entende-se a parte claramente isolada do fundo). A operação de afinamento é uma transformação homotópica na qual os componentes vão tendo a sua espessura reduzida até um número pequeno (usualmente 1 pixel), sem que haja mudança no número ou tipo de componentes. A definição de afinamento usa a transformada hit-miss.

$$V \textbf{ afi } X \;=\; X \;/\; (V \; him \; X)$$

onde a operação $X \;/\; Y$ deve ser entendida como $X \;\cap\; Y^c$. Veja-se um exemplo desta última operação. Se C_1 é a matriz

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	1	1	1	1	0	0
0	1	1	1	1	0	0
0	1	1	1	1	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

e a matriz C_2 é

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	1	1	0
0	0	0	1	1	1	0
0	0	0	1	1	1	0
0	0	0	0	0	0	0

então a operação C_1/C_2 será

```
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 1 1 1 0 0
0 1 1 0 0 0 0
0 1 1 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

O processo completo de afinar consiste em iterar a imagem com um ou vários pares de de elementos estruturantes até não haver mais modificações no resultado final. Afinar é um processo homotópico e portanto ele não altera a propriedade da conectividade. Quando o resultado final é alcançado a transformação vira idempotente e portanto continuar afinando não alterará o resultado alcançado.

A escolha do par estruturante é importante. A literatura (vide, por exemplo, FAC96, pág. 72) apresenta diversas sugestões de pares para afinar. Esta não é uma escolha trivial e os parâmetros para ela ainda não são completamente dominados.

Vejamos um exemplo de afinamento, na forma de uma matriz $a0$, com a máscara ma

```
ma = 0 1 1
     2 1 1
     2 2 0
```

Para a construção da máscara acima, usou-se uma codificação especial. A intenção é colocar na mesma máscara 2 conjuntos distintos, denominados de B^i e B^e, respectivamente os estruturantes internos e externos da imagem. Como ambos são conjuntos disjuntos isso pode ser feito. Usando a referência vista acima, temos:

conjunto	visto acima como	valor aqui
B^i	x	1
B^e	o	2
don't care	?	0

Portanto, a ma acima, representa o seguinte elemento estruturante

```
ma = ? x x
     o x x
     o o ?

     a0
0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 0 0
0 1 1 1 1 1 1 0 0
0 1 1 1 1 1 1 0 0
0 0 0 0 1 1 1 0 0
0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0
```

$a1 \leftarrow ma\ afinamento\ a0$

```
     a1
0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 0 0
```

 ©88-09, Pedro Kantek

```
0 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 0 0
0 0 0 0 1 1 1 0 0
0 0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0 0
```

$a2 \leftarrow ma\ afinamento\ a1$

```
      a2
0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 0 0
0 0 0 1 1 1 1 0 0
0 0 0 0 0 1 1 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0
```

$a3 \leftarrow ma\ afinamento\ a2$

```
      a3
0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 0 0
0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0
```

$a4 \leftarrow ma\ afinamento\ a3$

```
      a4
0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 0 0
0 0 0 0 1 1 1 0 0
0 0 0 0 0 1 1 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0
```

$a5 \leftarrow ma\ afinamento\ a4$

```
      a5
0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 0 0
0 0 0 0 0 1 1 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0
```

$a6 \leftarrow ma\ afinamento\ a5$

```
      a6
0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 0 0
0 0 0 0 0 0 1 0 0
```

```
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0
```

$a7 \leftarrow ma\,afinamento\,a6$

```
      a7
0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0
```

Note que $a6$ e $a7$ (e eventualmente todos os seguintes) são iguais. A transformação virou idempotente a partir deste ponto.

11.11.8 Espessamento

O espessamento, é definido a partir da transformada hit-miss como sendo

$$X\ \mathbf{esp}\ V = X\ \cup\ (V\ \mathbf{him}(X))$$

ou

$$X\ \mathbf{esp}\ V = X\ \cup\ (B^i\ \mathbf{ero}(X) \cap\ B^e\mathbf{ero}(X^c))$$

11.11.9 Esqueletização

A esqueletização é apenas a automação do procedimento de *afinação* visto acima, até que nenhum pixel a mais seja retirado da imagem anterior. No exemplo acima, é o processo que começa com $a0$ e termina com $a6$.

11.12 Segmentação por contorno ativo

Uma outra técnica moderna, introduzida por Kass em 1987 é a de uso de snakes para segmentar uma imagem. Um snake é um modelo deformável que irá sendo expandido e ajustado até se "grudar" com o contorno do objeto que se quer segmentar. Um snake é formado por um número (parametrizável) de snakels que se ligam entre sí, formando originalmente uma circunferência. A técnica sugere gerar o snake dentro do objeto a segmentar e ir expandindo-o, até que ele colida com o contorno real do objeto.

As forças que atuam sobre o snaxel são várias.

A energia externa atua ativada pelo gradiente da imagem. Quanto maior o gradiente de um ponto mais ele atrairá o snaxel. Com isso, o snaxel deve estacionar sobre os pontos de maior gradiente, supostamente os pontos da imagem que pertencem ao contorno do objeto a segmentar. Supondo δI o gradiente do ponto I da imagem, a energia externa é calculada como

$$E_{ext}(p_i) = -||\delta I||$$

A energia de continuidade funciona como se os snaxels estivessem ligados entre sí. Quando um snaxel é levado em uma direção, ele acaba puxando os snaxels vizinhos. Quanto maior a distância entre dois snaxels vizinhos, maior será a força entre eles.

 ©88-09, Pedro Kantek

Dados dois snaxels vizinhos, x e y, a energia de continuidade pode ser calculada a distância entre eles, como

$$E_{cont}(x) = ||p_x - p_y||^2$$

A força de curvatura tende a minimizar o número de pontos de curvatura possíveis. Esta energia penaliza o snake encurvado. Ela é calculada considerando-se 3 snaxels contíguos x, y e z e olhando-se a diferença entre os vetores formados por $x - y$ e $y - z$. Eis como fica a fórmula

$$(z - y) - (y - z) = ||z - 2y + x||^2$$

A função da energia O objetivo é ter o snake com energia mínima. Para tanto, deve-se minimizar tanto a energia externa como a interna, esta última formada pela soma entre a continuidade e a curvatura. Ela tem fórmula

$$E_{int}(y) = \alpha_y||y - z||^2 + \beta_y||z - 2y + x||^2$$

Os parâmetros α e β determinam a velocidade e a forma da convergência do snake sobre o contorno. Um valor pequeno de α relaxa as tensões entre os snaxels, permitindo que haja mais separação entre eles. Ainda que possa haver um α específico por snaxel, o usual é que o mesmo valor seja compartilhado por todos. Se o valor for elevado, os snaxels serão mais rígidos. O parâmetro β indica a curvatura que será permitida. Se a figura contém objetos irregulares com grande curvatura, deve-se usar um valor baixo para β.

A energia total do snake será igual à soma da energia de todos seus snaxels. E, ela deverá ser minimizada.

$$E(v) = \sum_{i=0}^{n-1} E_{int}(v_i) + E_{ext}(v_i)$$

Para minimizar a função de energia, deve-se usar um algoritmo de ajuste que vá movimentando os snaxels até suas posições ótimas. Cada snaxel terá uma vizinhança e deve-se buscar em que posição desta vizinhança a função de energia é mínima, para movermo-nos a essa posição. Aqui, pode-se usar o algoritmo guloso, que entretanto não garante a localização do mínimo global. Ele pode ficar preso em mínimos locais. A vantagem do método é sua rapidez, posto que sua complexidade é $O(n \times m)$ onde n é o número de iterações e m e o número de vizinhos considerados.

O problema aqui é se nenhuma vizinhança enxergar o contorno. Nesse caso, a força externa será zero e o snake tenderá a uma circunferência próximo (ou no ponto onde) começou. Diversas estratégias podem ser consideradas: o método da **bexiga**, no qual uma força radial atua sobre os snaxels até estes cruzarem o contorno, o método de **programação dinâmica** que força a busca em vizinhanças cada vez maiores, entre outros.

11.13 Uma aplicação real: AFIS

[2] Um sistema AFIS captura impressões digitais e permite processá-las estabelecendo um relacionamento entre as impressões e pessoas que tenham sido previamente cadastradas.

Existem duas maneiras de projetar um sistema AFIS. Na primeira, chamada **1:1**, cadastram-se pessoas que supostamente terão algum direito restrito a elas e suas impressões digitais. Posteriormente, confirma-se a presença dessas pessoas pela análise das digitais fornecidas.

[2] AFIS - Automated Fingerprint Identification System

É o caso de sistemas que usam impressões digitais como senhas. O nome 1:1 indica que o universo de pessoas a consultar é pequeno e para cada tentativa todo o banco de dados pode ser percorrido.

Os sistemas **1:n**, ao contrário, buscam identificar uma determinada impressão, a priori desconhecida, contra um arquivo de impressões, que em geral é grande, para se verificar se aquela impressão já não fora anteriormente capturada.

Existem 2 tipos de erro associados a este exame.

falso negativo O chamado falso negativo, quando a impressão já existe no arquivo, mas ela deixa de ser recuperada.

falso positivo O falso positivo, é o retorno de uma impressão de pessoa diferente como se fosse daquela proprietária da impressão sob análise.

Todos estes sistemas analisam a imagem da impressão, efetuam uma série de processamentos visando melhorar a imagem e depois identificam-na. Os processos de melhoria são o recorte, a limiarização e o afinamento. A identificação passa pela localização de pontos chave na impressão (denominadas minúcias) e sua codificação de alguma maneira.

Cada software em uso no mundo para AFIS usa uma tecnologia proprietária para fazer esta identificação. Por exemplo, no software ARID (Análise e Reconhecimento de Impressões Digitais) desenvolvido aqui pertinho de nós, no CEFET-PR e em uso em inúmeros clientes Brasil afora, a identificação se faz pela criação de um grafo descritor das minúcias.

A transformação da imagem em um descritor de minúcias, também conhecido como template, dá-se com uma razoável economia de espaço. Uma imagem de impressão digital (512 x 512 pixels em 256 níveis de cinza, em 500 dpi) que ocupa 262K bytes antes de ser comprimida e entre 20 e 30 KB após, gera um template de cerca de 5K bytes. O template não só é menor, como permite comparação e cálculo de similaridade, coisas que a imagem pura não permite.

O algoritmo de compressão utilizado neste tipo de sistema é conhecido como WSQ e garante taxas de compressão de cerca de 15:1. Além do template, necessário para posteriormente recuperar a identidade do proprietário de uma dada impressão, as imagens também têm de ser guardadas. Isto é feito para possibilitar a um perito humano a emissão de um laudo comparativo. De acordo com a lei, peritos têm de confrontar duas imagens antes de atestar pela igualdade e além disso, peritos não sabem interpretar templates.

Em resumo, um sistema AFIS de identificação de pessoas, passa pelos seguintes passos:

1. Cadastramento e identificação de pessoas, através de 2 fotos (frente e perfil), 10 imagens de impressão digital, dados alfanuméricos, assinatura e, eventualmente, imagens dos originais de documentos apresentados no cadastramento. A captura das digitais pode ser feita on-line (em sensores especiais) ou pelo rastreamento (scanner) de impressões de dedos entintados em papel, o que se conhece na gíria como tocar piano.

2. Para os 10 dedos capturados, as imagens são convertidas em WSQ e salvas.

3. Para cada imagem WSQ salva, o sistema AFIS faz um tratamento e uma extração de minúcias, gerando um template, que também é salvo.

Tudo isto é armazenado em computadores, à espera do dia em que uma certa imagem de uma digital, ou de parte de uma digital (também conhecida como latente) precise ter sua origem investigada. Por exemplo, na reidentificação de pessoas ou no levantamento de impressões na cena de um crime. Nestes casos:

 ©88-09, Pedro Kantek

1. A impressão é obtida diretamente da pessoa ou indiretamente através de técnicas policiais.

2. É fornecida ao sistema AFIS, junto com parâmetros de limiares de confiança.

3. O AFIS processa essa impressão, gera-lhe as minúcias e passa o template agora gerado para um computador que terá a missão de confrontar este template com os templates lá guardados. Este computador é denominado match server.

4. Todas as impressões existentes no arquivo serão colocadas em um dos 3 grupos:

   (a) Seguramente não é esta pessoa.

   (b) Talvez seja esta pessoa.

   (c) Seguramente é esta pessoa.

   O tamanho dos grupos é controlado pelo parâmetro limiar. Para um limiar pequeno, o grupo do talvez será pequeno (ou até vazio) e o grupo da certeza terá 1 ou nenhum elemento. O risco aqui é o do falso negativo.

   Para um limiar maior, o grupo do talvez cresce e, eventualmente, até o grupo da certeza. O problema aqui é a maior quantidade de trabalho manual na posterior confrontação de imagens.

Note-se que a busca, mesmo feita por computadores, é demorada, razão pela qual na medida em que cresce o tamanho do arquivo há que se colocar mais match servers a fim de manter o tempo de busca e recuperação dentro de parâmetros aceitáveis. Qualquer auxílio na busca é bem vindo. Por exemplo, se for dito que uma determinada impressão é de um dedo da mão direita, o tamanho da busca cai à metade. Se se conseguir individualizar de qual dedo é a impressão, a busca cai a 10% do tamanho original.

Apenas para exemplificar esta parte, vamos aos seguintes dados do sistema de identificação do DETRAN/PR. Lá existem dois problemas: garantir que uma mesma pessoa não tire duas carteiras distintas e garantir que na hora do exame o examinado seja de fato a pessoa em nome da qual sairá a carteira. O sistema tem um arquivo de 750.000 impressões digitais (cerca de 150.000 pessoas com 5 dedos em média de cada um) e um ritmo de entrada de cerca de 1000 novas pessoas/dia, são processadas quanto à duplicidade em cerca de 1h30 em um Pentium 450MHz com 2 processadores. O objetivo deste sistema é impedir que pessoas já cadastradas busquem um novo cadastro falseando informações de identidade. Ressalte-se que o sistema já pegou uns bons quantos sujeitos tentando fazer isso.

Outro exemplo bem interessante de uso desta tecnologia está na Secretaria de Segurança Pública. Lá um sistema AFIS está identificando e cadastrando todas as pessoas que passam pelo centro de triagem. Ressalte-se que no aspecto criminal é muito importante que as pessoas sejam identificadas corretamente. O sistema já está em produção há diversos meses e tem apresentado resultados mais do que animadores.

Existem diversos padrões regulando esta área, todos derivados do padrão ANSI / NIST ITL 1.2000. Os principais são o EFTS (implementação do FBI), a Interpol Implemenentation, a United Kingdom Implementation e a norma ASPID. Embora templates gerados em um sistema não possam ser confrontados com templates gerados em outros, as imagens de impressões podem. Estas normas, entre outras coisas estabelecem como as imagens devem ser intercambiadas, a fim de que pessoas possam ser identificadas em outros sistemas.

Para saber mais, consulte os sites www.antheus.com.br e www.pequi.com.br

11.14 Análise de Imagens

Depois que a imagem já foi processada usando técnicas vistas em aulas passadas, pode-se passar à fase de análise da imagem, na qual características da cena são extraídas e usadas para a tomada de decisão. Usualmente, como uma única imagem pode corresponder à diversas cenas (base de todas as ilusões de ótica), usualmente são necessárias ou imagens adicionais da mesma cena, ou informações não visuais adicionais, ou ambas as coisas.

Este conhecimento pode ser específico ou genérico e pode assumir inúmeras formas, não estando limitado de nenhuma maneira. (O que está na cena, onde estão as fontes de iluminação, quais as fontes de ruído e o seu valor, a cena é estática ou dinâmica, os entes são autônomos ou não e de que tipo, objetos regulares, irregulares, côncavos, convexos, lineares, cônicos, cores, brilhância, refletância (albedo), existência de texels, etc etc. Aqui vale sempre a regra: quanto mais genérico for o dispositivo analisador, maior sera a demanda por informações adicionais. O reverso é verdade: quanto mais específico for o analisador menos informação é necessária.

11.14.1 exemplo: reconhecendo placas

Um exemplo do que se disse acima, é dado pelo reconhecedor de placas de veículos de trânsito. Colhida a foto, a mesma precisa sofrer diversas operações, antes de permitir a identificação da placa do veículo.

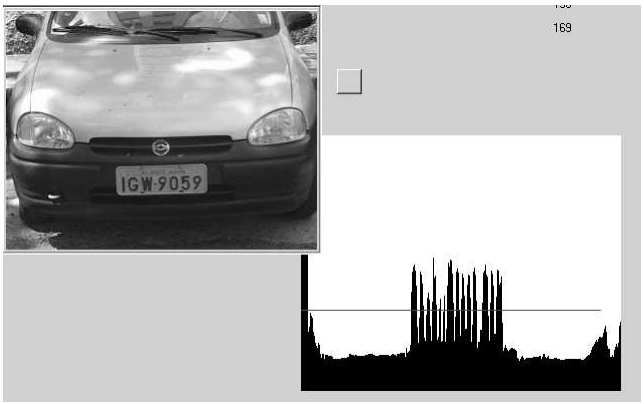
- captura da foto e seu registro em meio digital
- eliminação de ruído e equalização de condições (fotos diurnas $\times$ noturnas, em dia de sol $\times$ em dia de chuva, etc)
- limiarização da foto
- Localização do retângulo definidor da placa (lembrar que este retângulo pode estar em diversas posições dentro da foto)
- eventual necessidade de rotação
- esqueletização da imagem
- alimentação de uma rede neural
- a saída da rede neural informa os 7 caracteres formadores da placa

Para o algoritmo de localização do retângulo definidor, um algoritmo muito usado é o da busca de assinaturas. Para o caso de uma placa, a assinatura é de alguns pixels escuros sobre um fundo mais claro. Para buscar esta assinatura pode-se construir um gráfico:

1. cada linha da imagem original dá origem a um gráfico diferentes
2. a abscissa do gráfico corresponde à abscissa do pixel na imagem
3. a ordenada corresponde ao nivel de cinza do pixel. Valores baixos correspondem a preto e altos a branco

Note-se que quando a linha guia atravessa a placa, aparece um serrilhado característico no gráfico. Alguns pixel antes do serrilhado aparecer, e alguns depois constituem os limites horizontais do retângulo. Nas linhas guias, alguns pixels antes e alguns depois do serrilhado, constituem os limites verticais do retângulo. Um exemplo desta estratégia é dada pelo programa: reconhecedorpr.exe. Neste a linha vermelha constitui a média ideal para buscar a assinatura (que varia de foto a foto).

 ©88-09, Pedro Kantek



11.14.2 Mundo dos cubos

Na abordagem desta aula, usa-se a segunda estratégia, considerando características do mundo físico que limitam as possíveis coisas que estão em cena. Usaremos aqui imagens de um mundo particular sujeito às seguintes restrições:

- Haverá apenas poliedros em cena (faces planas e arestas retilíneas)
- A iluminação não provocará sombra
- os poliedros não contém fendas
- em qualquer ponto nunca se interceptam mais do que 3 superfícies.

Este tipo de cena recebe o nome de *poliedros com vértices triédricos* e no que nos importa, gera imagens compostas apenas de linhas retas.

Na imagem formada, os planos podem se interceptar apenas e somente apenas de 3 maneiras: Um tipo especial de aresta que conecta planos que na realidade não tem conexão física, chamada aresta de oclusão (Winston a chama de linha de contorno). Neste tipo de aresta, apenas um dos planos é totalmente visível, o outro está escondido.

Deve-se marcar este tipo de aresta na imagem usando uma flecha `(->-)` e o sentido da flecha é tal que o plano ocultado está à esquerda da flecha. Winston diz que o objeto contornado deve ficar à direita da flecha, o que vem a ser a mesma coisa.

Para as arestas que conectam dois planos com ambos visíveis, existem dois casos: as arestas convexas (identificadas com um sinal de $+$) ou côncavas, estas identificadas por um sinal de $-$. Procure lembrar que uma caixa de fósforos sobre a mesa terá arestas convexas enquanto a aresta que conecta a parede e o piso da sala onde estamos é uma arestas côncava.

As aresta côncavas indicam arestas que conectam 2 faces em ângulos menores que 180 graus relativamente ao observador. Já as arestas convexas identificam faces que se conectam em ângulos maiores que 180, do ponto de vista do observador.

Dewdney, chega a sugerir um quarto tipo de aresta a que ele chama de *crack* que conecta partes do mesmo plano (como se fosse um risco reto a caneta feito sobre uma parede branca). Esta consideração não será usada aqui.

A questão chave aqui é que o mundo físico dos poliedros impõe certas restrições para que a imagem exista na realidade. Levando estas restrições adiante, em quase todos os casos será possível rotular corretamente todas as arestas da imagem, e consequentemente será possível inferir características físicas da cena representada na imagem.

Os casos exepcionais ficam por conta dos poucos casos em que a ambigüidade presente impedir a correta análise da imagem. Neste caso, novas imagens (com variações

sobre o ponto de tomada da imagem) ou mais informações adicionais da cena serão necessárias. Hoffman (Hof00) descreve com bastante profundidade e amplidão a eliminação da ambigüidade através de mudanças de ponto de vista e vale a pena ser consultado.

Antes de começar a marcar as arestas e a propagar seus efeitos, uma etapa necessária é rotular as junções de arestas segundo 4 categorias (Dewdney sugere 5), que recebem o nome de V, W, Y e T. Deve-se ressaltar que TODAS as junções possíveis estão aqui representadas. Impressiona o número baixo de junções possíveis.

Acompanhe o desenho

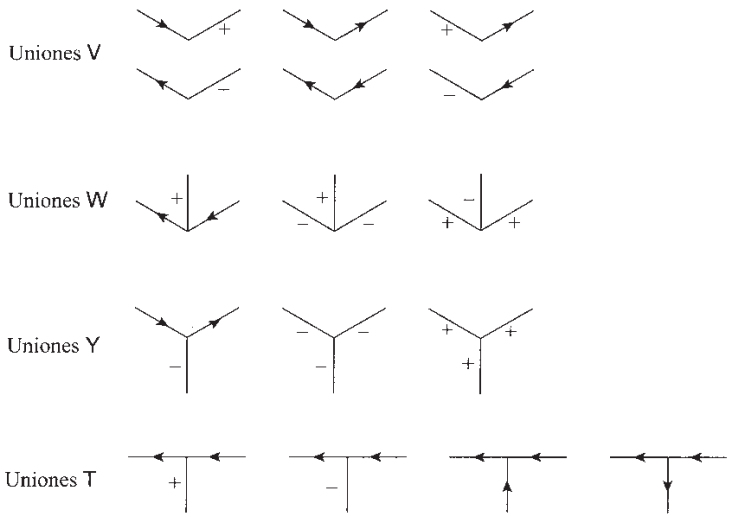


Figura 11.65: Quatro tipos de junções

Vamos procurar identificar o que caracteriza cada junção

junção	quantas	característica
V	6	é a única junção que apresenta apenas 2 linhas
W	3	3 linhas incluídas em menos de 180^o
Y	3	3 linhas que ocupam mais de 180^o
T	4	3 linhas, sendo duas delas colineares

Depois de ter marcado todas as junções, pode-se passar à segunda fase do algoritmo que busca identificar as arestas. Antes de prosseguir, tente identificar as junções que aparecem neste desenho representando uma sala com um corredor ao fundo e tendo um cubo colocado no meio dela.

A resposta ao exercício é

Para identificar as arestas, podem-se usar as seguintes heurísticas:

- Ao examinar a imagem, suporemos que todos os objetos estão suspensos no espaço. Assim cada aresta do fundo do objeto (o contorno) tem apenas arestas flecha.
- Note que há apenas uma junção "W" que tem flechas em suas pontas. Assim, a outra aresta é um "+".
- Há apenas um "Y" que recebe "+" em uma aresta. Ocorrendo isso, todas as demais arestas são também "+".

  Finalmente, as restrições vão sendo repassadas adiante, de modo que uma aresta originalmente rotulada em uma extremidade, deve obviamente apresentar o mesmo rótulo na outra extremidade. Afinal, são objetos físicos.

 ©88-09, Pedro Kantek

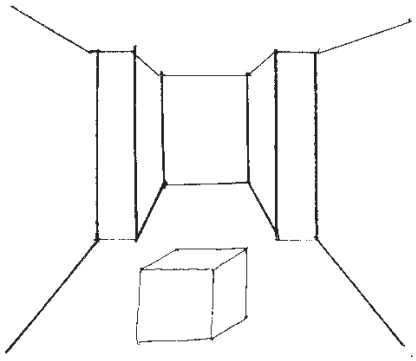


Figura 11.66: exercício

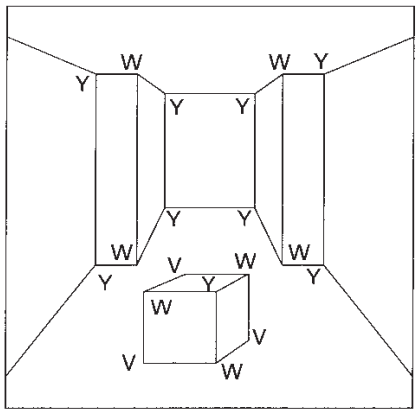


Figura 11.67: resposta ao exercício

Os 4 tipos mostrados não esgotam o assunto, há mais (junções em K, em X, em pico e em psi), mas estas podem ser abandonadas se forem feitas 4 hipóteses razoáveis:

1. Não existem sombras
2. Não existem fendas
3. Vértices limitados a triedros. Não é possível ter um vértice com mais de 3 superfícies (o topo da Pirâmide de Keops está excluído).
4. O ponto de observação é geral. Nenhuma junção muda de tipo com uma pequena mudança de posição do olho. Por exemplo, imagine um cubo sendo olhado de frente.

Finalmente, perceba-se que se uma figura "maluca"como a mostrada em 11.79 for mostrada ao programa, ele não consegue apresentar nenhum resultado. De fato, tal figura é fisicamente impossível.

Posteriormente, Waltz incluiu na análise fendas e iluminação. Se isso torna o problema bem mais complexo, ele também afasta a implementação de um "toy domain" aproximando-o de aplicações reais.

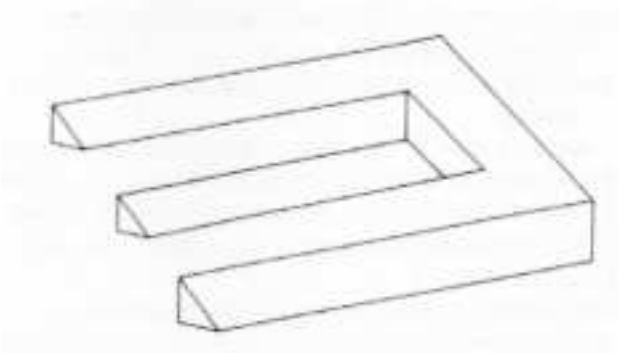


Figura 11.68: Confundindo o programa

A iluminação (inicialmente por um foco de luz único) traz sombras. Estas podem ser de 3 tipos: Uma face pode estar iluminada, pode estar na sombra de outro objeto (que esta entre a face e o foco de luz) ou finalmente pode estar volta em direção distinta da que vem a luz (isto é chamado auto-sombreamento).

Agora são 11 as possibilidades de identificação de linhas, e com o acréscimo de 3 tipos de sombra, poderia haver 99 tipos de linha ($3^2 = 9 \times 11 = 99$), mas somente cerca de 50 dessas 99 linhas são fisicamente possíveis.

O programa de Waltz teve sucesso em analisar este tipo de imagem, mostrado na figura 11.69

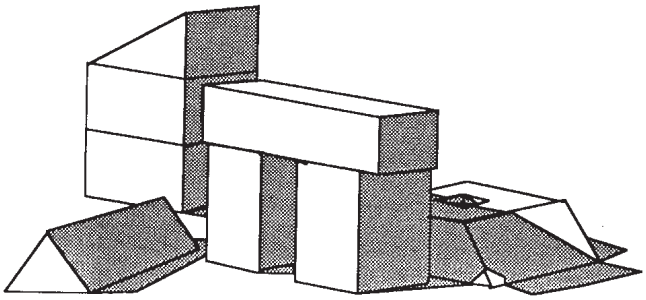


Figura 11.69: identificando cenas mais complexas

11.15 Exercícios

1. Etiquete corretamente a figura 11.70.
2. Demonstre a impossibilidade física da construção do triângulo do diabo, mostrado na figura 11.71.

Desafio

- Implementar este algoritmo de análise de bloquinhos.

OBSERVACAO: a seguir a antiga aula IM4, que foi substituída pelo que está acima. Mantive este texto aqui embaixo para não perder o esforço.

 ©88-09, Pedro Kantek

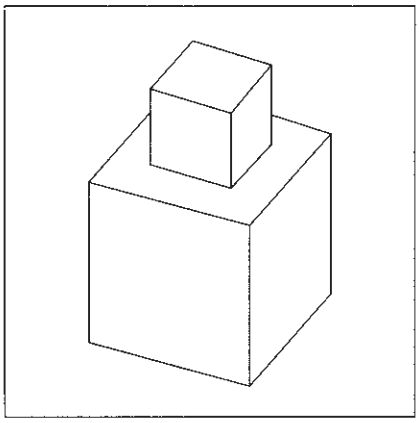


Figura 11.70: etiquete

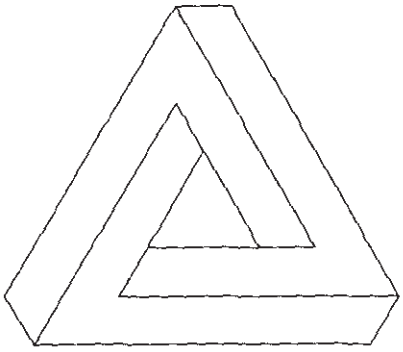
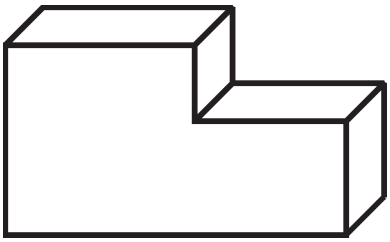


Figura 11.71: Construção impossível ?

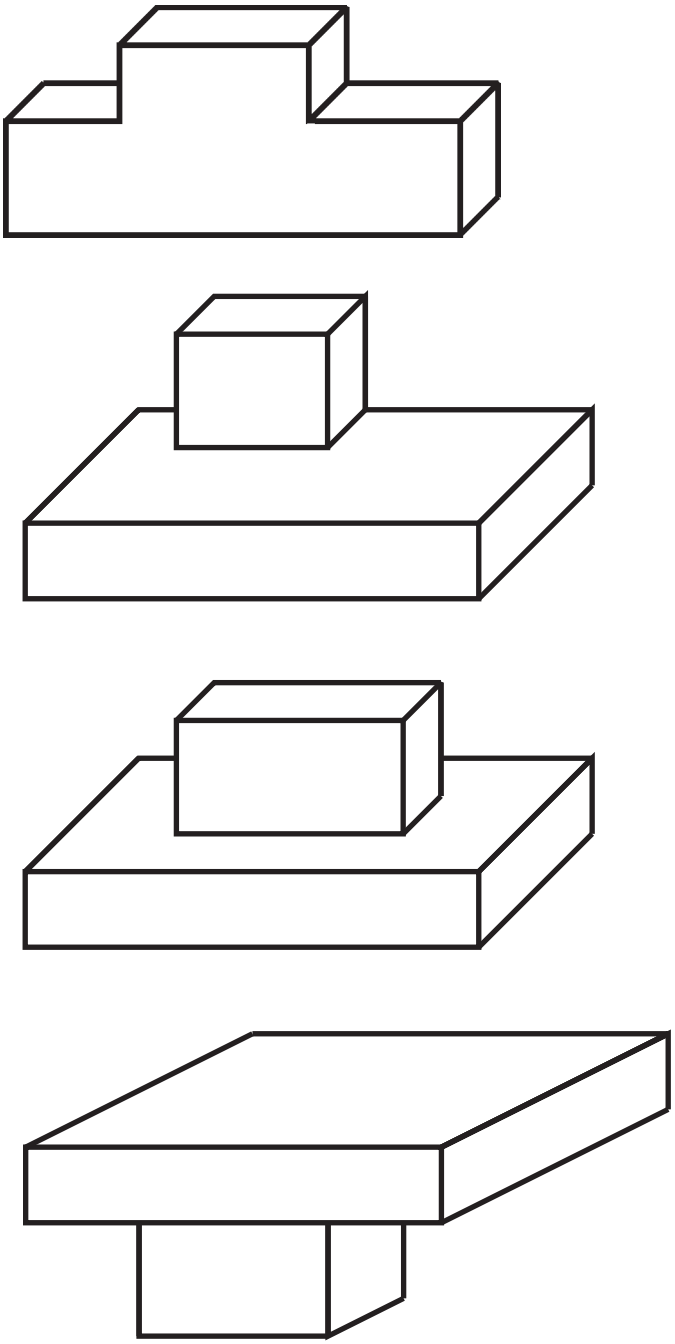


imagens do exercício

Nas imagens deste mundo, as linhas se conectam por apenas 5 tipos, segundo a descoberta de David Huffman e Maxwell Clowes, estendida depois por David Waltz no MIT em meados dos anos 70.

Suponhamos um mundo de poliedros, devidamente iluminados para não formarem sombras. Ao olhar para um desenho como

nosso olho (e cérebro) é capaz de determinar existirem 5 blocos distintos no desenho: as duas plataformas horizontais, duas colunas verticais e a haste horizontal superior.



Embora não se saiba como o cérebro analisa este tipo de imagem, existem evidências experimentais que sugerem que o cérebro siga um tipo de análise e raciocínio similar a este.

Os 5 tipos são: Flecha, garfo, L, T e K. Veja no desenho

Para analisar a imagem a partir destes tipos, há necessidade de rotular as arestas. Por enquanto, há necessidade de usar apenas 4 rótulos nas arestas e cracks.

Aresta côncava: identificada por um sinal de - (menos) ela indica arestas que conectam 2 faces em ângulos menores que 180 graus relativamente ao observador. A aresta

 ©88-09, Pedro Kantek

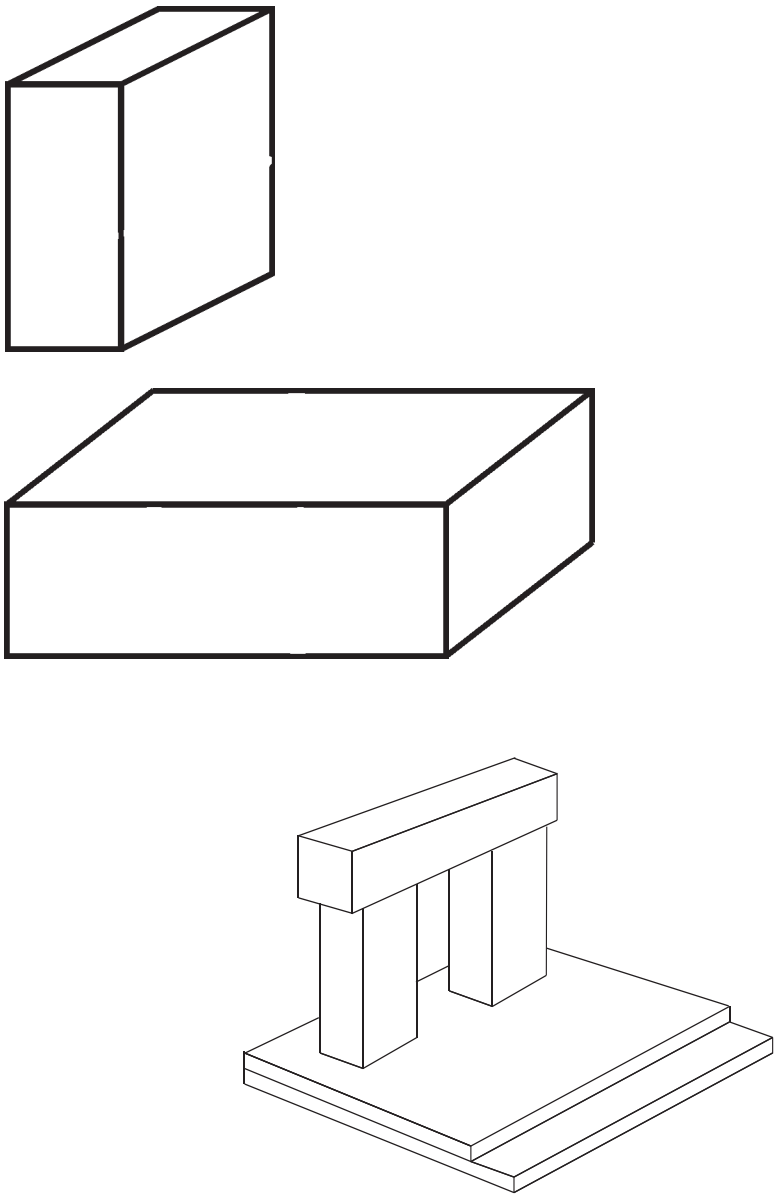


Figura 11.72: Uma construção de bloquinhos

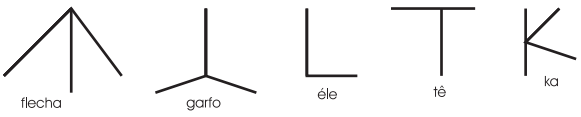


Figura 11.73: Cinco tipos de conexões

que conecta a parede e o teto desta sala é um exemplo de aresta côncava.

Aresta convexa: identificada por um sinal de + (mais), ela indica faces que se conectam em ângulos maiores que 180, do ponto de vista do observador. Se olharmos a

caixinha de giz sobre a mesa, as arestas da caixa são convexas.

Crack: representando pela letra C, é uma aresta que conecta duas superfícies que formam ângulo de 180 graus. Na verdade é como um risco sobre uma superfície qualquer.

Aresta de contorno: aquelas que conectam duas regiões que não se tocam. Elas contornam o objeto em análise isolando o que faz parte e o que não faz parte da análise. Estas arestas são identificadas por flechas colocadas no meio da aresta. Por exemplo —->——. A direção da flecha é estabelecia fazendo-se um passeio ao longo da linha de contorno, mantendo-se o objeto contornado à direita. A direção do passeio é a direção da flecha.

Figura 11.74: Rotulando arestas

Usando estas arestas, podem ser construídas as junções possíveis em desenhos de poliedros. As possíveis estão representadas na figura 11.75. Surpreendentemente, a lista não é grande. Restrições de ordem física fazem com que poucas junções sejam possíveis. O ser humano consegue rotular as arestas e junções de arestas a partir do conhecimento implícito da realidade física da imagem. O objetivo agora é o inverso. Um programa de computador deve inferir a realidade física subjacente à imagem a partir do conhecimento das arestas e das junções.

Os 5 tipos mostrados não esgotam o assunto, há mais (junções em X, em pico e em psi), mas estas podem ser abandonadas se forem feitas 4 hipóteses razoáveis:

1. Não existem sobras
2. Não existem fendas
3. Vértices limitados a triedros. Não é possível ter um vértice com mais de 3 superfícies (o topo da Pirâmide de Keops está excluído).
4. O ponto de observação é geral. Nenhuma junção muda de tipo com uma pequena mudança de posição do olho. Por exemplo, imagine um cubo sendo olhado de frente.

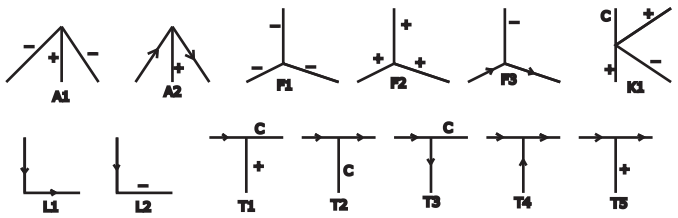


Figura 11.75: Conexões possíveis

Estas são todas as possibilidades de conexão.

Na figura 11.76, tem-se a figura original, na qual escolheu-se um junção, marcada no desenho com uma flecha. Prossegue-se por todas as linhas, marcando todas as junções de acordo com a tabela anterior. Não importa por qual junção se começa, o programa de Waltz gera sempre o mesmo resultado. Estabelecido o início, o programa rotula

 ©88-09, Pedro Kantek

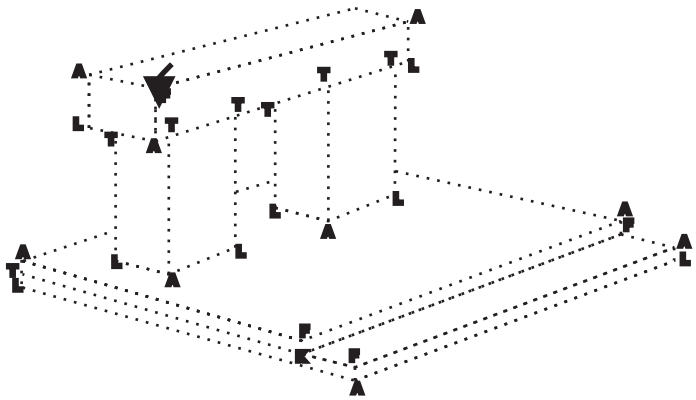


Figura 11.76: Uma possível "visualização"

as arestas e compara cada junção com seus vizinhos. Dessa análise, e por exclusão, o programa consegue individualizar os poliedros que compõe a imagem. veja-se na figura 11.77 um detalhe de como é feita essa análise. Já na figura 11.78 o resultado final da análise.

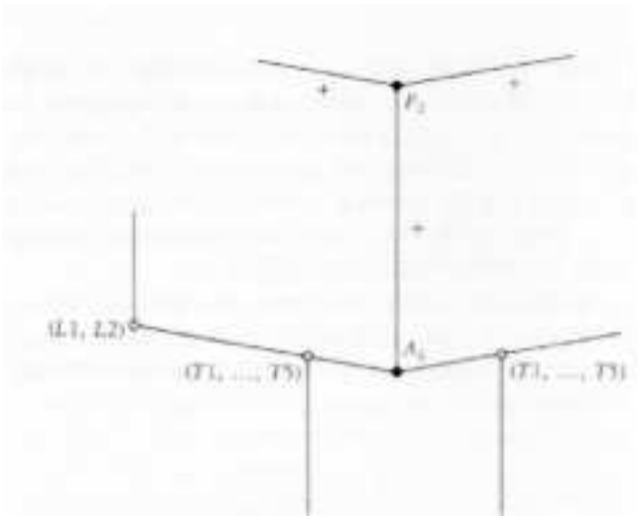


Figura 11.77: Em detalhe

Finalmente, perceba-se que se uma figura "maluca"como a mostrada em 11.79 for mostrada ao programa, ele não consegue apresentar nenhum resultado. De fato, tal figura é fisicamente impossível.

Desafio

- Implementar este algoritmo de análise de bloquinhos.

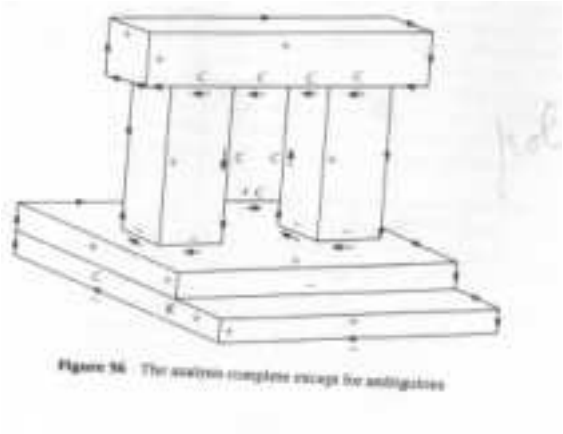


Figura 11.78: Sentidos de busca

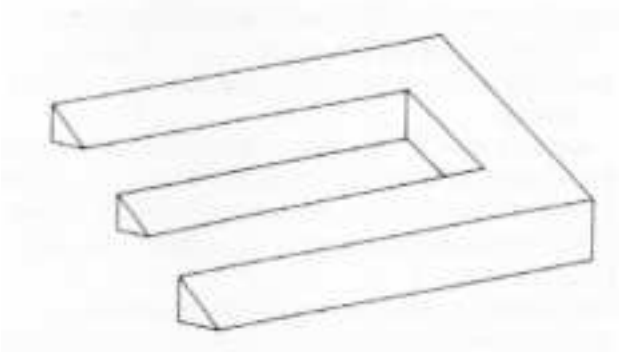


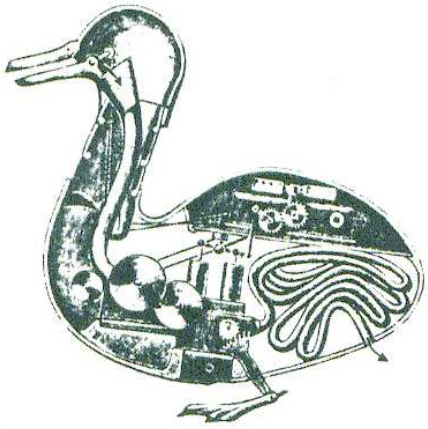
Figura 11.79: Confundindo o programa

 ©88-09, Pedro Kantek

Capítulo 12

Robótica

Um **robot** (ou como alguns preferem, robô), é um agente físico que executa uma tarefa manipulando o mundo físico. O nome robot foi usado pela primeira vez pelo dramaturgo tchecoslovaco Karel Kapek em peça teatral escrita por ele em 1921, de nome R.U.R. (Rossum's Universal Robots). Na peça, os robôs não eram construídos, eles cresciam quimicamente e ressentidos com seus mestres humanos, decidem assumir o comando. Parece, entretanto que o inventor da palavra foi o irmão de Karel, Josef que, em uma breve narrativa de 1917 (*Opilec*), combinou as palavras *robota* (trabalho obrigatório) com *robotnik* (servo) para compôr a palavra robot. Já a expressão **robótica** foi usada pela primeira vez por Isaac Asimov, que embora cientista, ganhou fama como escritor de ficção científica. São dele, as 3 leis fundamentais da robótica. [1] Entretanto, a história de dispositivos desse tipo é bem mais antiga. Em 1738, Jacques Vaucanson construiu um pato para uma exposição em Paris. O pato mexia, nadava, mexia as asas e alisava as penas com o bico. Podia beber e pouco depois de haver "comido" evacuava uma matéria amorfa.



No sentido atual do termo um robô é um agente físico que executa tarefas manipulando o mundo físico. Para isso, eles são equipados com efetuadores (pernas, rodas, articulações e garras). Também têm sensores que lhes permitem perceber o ambiente (câmeras, emissores/receptores de ultra som, giroscópios, acelerômetros, etc).

[1] 1.O robô não atentará contra seres humanos, ou por inanição permitirá que eles sejam feridos; 2.O robô obedecerá aos comandos humanos, desde que eles não atentem contra a primeira lei; 3. O robô protegerá a sua própria existência, desde que isto não contradiga as leis 1 e 2.

Via de regra, os robôs se enquadram em 4 categorias: os manipuladores (robôs fixos), com uma cadeia de manipuladores controláveis, permitindo a atuação dos autuadores em qualquer posição dentro do local de trabalho. Existem mais de 1.000.000 de robôs deste tipo instalados no mundo. (Dado de 2003).

O segundo é o robô móvel. Eles se deslocam pelo seu ambiente usando rodas, pernas, lagartas ou dispositivos similares. A ênfase está no sensoreamento remoto. Por exemplo, o robô Sojourner enviado a Marte em julho de 1997, que pode ser visto na figura 12.1.

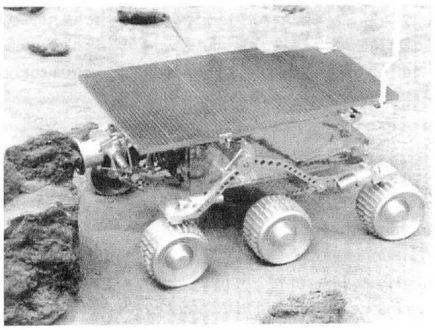


Figura 12.1: Robot Sojourner em Marte

O terceiro tipo é uma mistura dos dois anteriores. É móvel e tem manipuladores. Na figura 12.2 vê-se dois humanóides fabricados pela Honda do Japão.

Este tipo de robô tem atuação mais ampla que os fixos, mas sofre de duas dificuldades significativas: a complexidade do seu controle e a ausência de um ponto fixo rígido para aplicar forças.

Além destes tipos, há um quarto, bem amplo, que congrega: Próteses: membros artificiais, olhos e orelhas eletrônicas etc. Ambientes inteligentes, por exemplo, uma casa com sensores e efetuadores e Trabalhos cooperativos: Sistemas com diversos corpos, nos quais a ação robótica é alcançada por enxames de robôs cooperativos.

Dificuldades

As dificuldades no campo da robótica são bem consideráveis: Em primeiro lugar, o sensoreamento do robô é parcial (robôs não enxergam através de obstáculos) e sempre sujeito ao ruído. Além disso, o mundo real opera em tempo real, inviabilizando operações

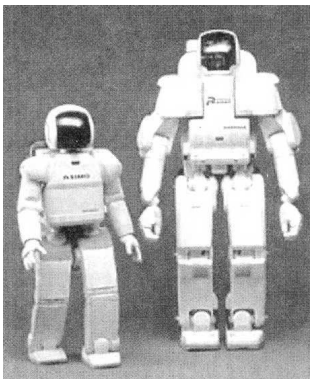


Figura 12.2: Humanóides da Honda

demoradas (uma bonita exceção aqui é o controle dos robôs marcianos. Há um retardo de 15 min entre uma ação e sua reação). Os comandos emitidos pressupõe incerteza, pois pode haver deslizamento de engrenagens, fricção, obstáculos, o que exige a obtenção de feed-back. Mais ainda, ao contrário do mundo virtual, aqui as colisões de fato causam prejuizos. Finalmente, uma dificuldade de ordem superior talvez seja atribuir o controle total ao robô, sem interferência humana.

12.0.1 Sensores

Há dois tipos de sensores: os passivos captam sinais gerados por outras fontes (uma câmera de vídeo, por exemplo). Os ativos enviam energia ao ambiente e recebem o resultado desse envio, por exemplo o sonar. Estes tendem a oferecer mais informação que aqueles, mas ao custo de consumir mais energia. Há ainda o perigo da interferência, quando mais de um robô opera no mesmo ambiente. Há 3 categorias de sensores: medidores de distância (telêmetros), capturadores de imagens, o que pressupõe seu tratamento posterior, e finalmente os sensores que medem propriedades do próprio robô.

Os telêmetros medem a distância do robô até objetos. Podem ser sonar, radar e laser. Quanto à distância de atuação podem ser táteis para pequenas distâncias ou – no outro extremo do espectro – sensores de GPS que medem distâncias até satélites que emitem sinais pulsantes. [2] Analisando a demora de receber os sinais de 3 satélites (existem duas dúzias deles em óbita) pode-se individualizar 2 pontos no espaço. Como um deles usualmente cai muito longe da superfície, pode ser ignorado. De uso livre, pode entretanto pode ser anulado em regiões de guerra (recentemente na região do Iraque) e não funciona em lugares fechados ou embaixo d´água.

Os sensores de imagem apresentam toda a dificuldade de análise de imagens. Pode-se usar aqui a visão estereoscópica para estimar distâncias.

Os sensores proprioceptivos informam ao robô seu estado. Podem ser decodificadores de eixos, sensores de odometria, sensores inerciais como giroscópios, sensores de força e de torque. Estes devem poder medir forças em 3 direções de translação e em 3 direções de rotação.

12.0.2 Efetuadores

Comecemos definindo grau de liberdade. Conta-se 1 GL para cada direção independente em que um robô ou seus efetuadores pode se mover. Por exemplo, um robô rígido (um braço) pode ter 3 GL para sua posição x, y, z e mais 3 para sua orientação angular conhecidos como yaw, roll e pitch. Estes 6 GL definem a pose do robô. O estado dinâmico dele inclui uma dimensão adicional para a taxa de mudança de cada dimensão estática.

12.1 Grand Challenge

O governo americano pretende ter um terço de suas forças armadas automatizadas até 2015. Assim, a DARPA (Defense Advanced Research Projects Agency) lançou em fevereiro de 2003 uma corrida de robots autônomos. A corrida foi inédita pelas dificuldades e tamanho. Ela ocorreu pela primeira vez em 13 de março de de 2004, e a sua segunda versão ocorreu a 8 de outubro de 2005. Trata-se de uma corrida entre robots, que devem de forma autônoma percorrer uma distância de 322 kilometros no deserto perto de Los Angeles, EUA. O caminho passar por todo tipo de terreno e o agente deve se guiar por 2 tipos de sinal: Uma planilha de pontos através de suas coordenadas terrestres vis a

[2] Vide a respeito uma boa explicação sobre o funcionamento do GPS em Scientific American Brasil, ano 3, número 25 de junho de 2004

 ©88-09, Pedro Kantek

vis leituras obtidas de um satélite GPS e de uma inspeção local (visual ? radar ? sonar ?) para o ajuste fino do trajeto. Na versão de 2004, nenhum veículo chegou perto, o que foi mais longe andou 7,5 milhas antes de desistir.

O prêmio na versão 2004 era de US$ 1.000.000 para o veículo que demorasse menos tempo, com limite máximo de 10h. Para 2005 este valor foi dobrado: agora é de US$ 2.000.000. A competição está aberta a cidadãos e organizações americanas, incluindo universidades e empresas. Mais de 100 times se inscreveram para a versão 04. 195 equipes se increveram na versão 05.

12.1.1 Regras

- O veículo deve cumprir o trajeto em no máximo 10 horas.
- O veículo deve se manter dentro do trajeto (uma estrada virtual de largura variável) fornecida pela DARPA em um CD horas antes da largada.
- O veículo pode usar sinais de GPS ou outros sinais públicos
- Nenhum comando de controle externo pode ser enviado ao veículo durante o trajeto
- O veículo não pode intencionalmente tocar qualquer competidor
- Uma estação de verificação e ajuste é permitida em um checkpoint localizado aproximadamente na metade do percurso.
- A corrida começa com disposição escalonada, um veneto de qualificação determina quem sai antes. Se senhum veículo ganhasse em 2004, a corrida seria repetida anualmente, até alguém ganhar ou a verba acabar (após 2007).

12.1.2 O percurso

Duas horas antes da partida, um funcionário da DARPA entrega a cada equipe um CD-ROM contendo uma série de coordenadas de GPS chamadas *waypoints* com espaço entre cada um variando entre 36 e 305 metros. A largura da rota também é variável: entre 3 e algumas centenas de metros, dependendo do trecho.

12.1.3 O algoritmo do sandstorm

A equipe entregou ao robot um plano de navegação bem detalhado. Mas, como o percurso final só será conhecido 2h antes da partida, a equipe reuniu mapas, modelos e imagens aéreas de toda a área potencial. A equipe obteve do US Geological Survey fotos aéreas e perfis tridimensionais da área, com capacidade de reconhecimento de objetos de até 1m. Depois, eles juntaram os mapas em um banco de dados geográficos que chegou a ter vários terabytes. Um programa usa este DB para calcular quanto custa atravessar cada metro quadrado da região. Algumas áreas como penhascos ou limites de percurso têm custos infinito, porque estragam o veículo ou desqualificam a equipe. Leitos de lagos secos, por exemplo, podem têr custo zero.

No dia da corrida, os dados reais do percurso serão transmitidos por um link de satélite de alta velocidade ao centro de controle. Lá um conjunto de computadores usará o mapa de custo para calcular a melhor rota. Uma dezena de voluntários treinados dividirão a rota em seções e revisarão os cálculos a fim de evitar que algum erro conduza o sandstorm ao desastre. Feito isso, a rota será transmitida ao robot pouco antes da partida.

Na figura 12.5 pode-se ver a versão 04 do Sandstorm. Na figura 12.3 qual a visão que ele enxerga a partir de seus múltiplos "olhos" e finalmente na figura 12.4 como ele mapeia o espaço e anda por ele.

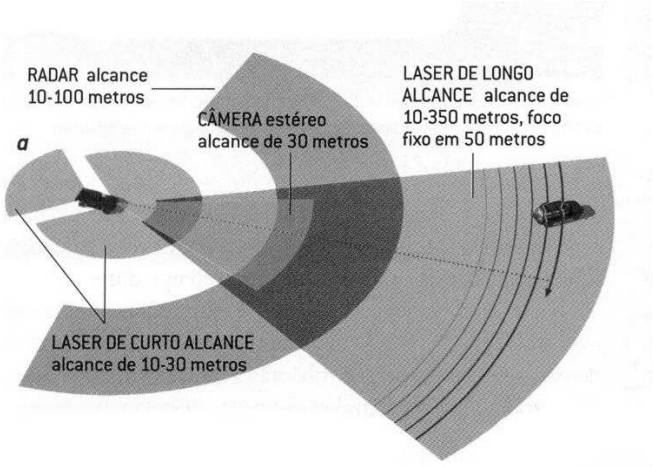
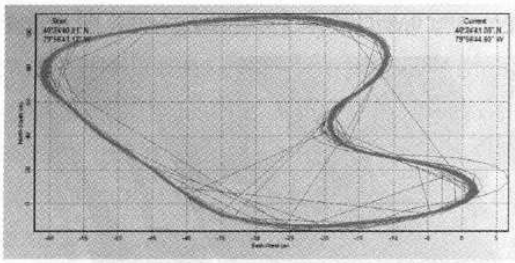


Figura 12.3: Os olhos do Sandstorm

1. RASTREAMENTO DE LOCALIZAÇÃO
Um computador de navegação Applanix usa dois receptores de GPS, três giroscópios de fibra óptica, três acelerômetros de silício e dois odômetros ultraprecisos para fixar a posição do robô, com variação de 50 cm, e medir sua orientação no espaço dentro de 0,4 grau. O robô recebe atualizações de sua posição 200 vezes por segundo.



Rota do GPS no teste de 191 km (linhas vermelhas são falhas dos sensores)

Figura 12.4: O mapeamento do espaço

12.1.4 Versão 04

Na versão 04, apenas 15 veículos estiveram alinhados para a largada. Eis o que aconteceu a cada um deles:

22-Red Team Na milha 7.4, saiu da estrada, bateu e um pneu dianteiro pegou fogo, que foi logo extinto. O veículo foi desligado externamente.

21-SciAutonics II Na milha 6.7, saiu da estrada em direção a um banco de areia. Atolou. Foi desligado.

7-Digital and Drive Na milha 6.0 o veículo bateu numa rocha do tamanho de uma

 ©88-09, Pedro Kantek

bola de futebol. Parou. Foi desligado.

5-Caltech Na milha 1.3, o veículo saiu da estrada, foi para a grama, voltou para a estrada e voltou para a grama. Foi desligado.

25-Virginia Tech Na área de largada o veículo travou os freios. Foi retirado.

23-Axion Race Veículo começou a andar em círculos na área de saída. Foi retirado.

2-CajunBot Ainda na área de saída, bateu num muro. Foi retirado.

13-ENSCO Veículo se movia sem muita direção, na milha 0.2 ao fazer uma curva de 90^o, tombou. Foi removido.

4-CIMAR Na milha 0.45, o veículo caiu numa valeta. Foi desligado.

10-Palos Verdes High School Bateu num muro na área de aprtida. Foi retirado.

17-SciAutonics I Na milha 0.75 o veículo saiu da rota. Tentou voltar a ela por 90 min. Sem nenhum movimento, o veículo foi desligado.

20-TerraMax Várias vezes, o veículo detectou diversos arbustos, próximos à estrada e desviou deles, retornando após. Na milha 1.2 não conseguiu retornar. Foi desligado.

15-TerraHawk Não conseguiu largar.

9-Golem Na milha 5.2 quando tentava subir um morrinho, ele não conseguiu. Após esperar 50 min ele foi desligado.

16-Blue Team Não conseguiu largar.

12.1.5 Versão 05

Dos 195 competidores, inicialmente 40 foram escolhidos para participar da qualificação. Três equipes foram adicionadas posteriormente (23/ago/05), perfazendo 43 equipes. A qualificação ocorreu entre 27/set e 5 de outubro.

No dia 6 de outubro, os robots foram transportados para o local da largada. O dia 7 foi reservado para ajustes e consertos.

No dia 8, cada robot largou em instantes de tempo separados. Todos eles foram parados em diversos momentos durante a corrida. Tais pausas foram debitadas do tempo final oficial. O prêmio só foi oficialmente assinalado no dia 9. Os competidores em 2005 tiveram muito mais sucesso do que os de 2004. Só um dos 23 competidores deixou de ultrapassar a melhor marca de 2004 (7.3 milhas). Ao final, 18 robots pararam no caminho e 5 deles chegaram ao final. Stanley (Universidade de Stanford), Sandstorm e Highlander (ambos da Universidade Carnegie Melon) concluiram com poucos minutos de diferença. O vencedor, Stanley, fez o percurso em 6h 53min 8seg com uma velocidade média de 19.1 MPH, Sandstorm demorou 7h 4min 50seg, com velocidade de 18.6 MPH.

Eis o resultado final

Stanley Stanford Racing Team,Stanford U., Palo Alto, Calif. 6:54, primeiro lugar

Sandstorm Red Team Carnegie Mellon U., Pittsburgh, PA 7:05, segundo lugar.

H1ghlander Red Team Too Carnegie Mellon U., Pittsburgh, PA 7:14 terceiro lugar

Kat-5 Team Gray The Gray Insurance Company, Metairie, La. 7:30 quarto lugar

TerraMax Team TerraMax Oshkosh Truck Company, Oshkosh, Wisconsin 12:51 Ultrapassou as 10 horas.

DEXTER Team ENSCO ENSCO, Inc., Springfield, Va. Parou na milha 81, devido a um incêndio no pneu

Spirit Axion Racing Westlake Village, Calif. Parou na milha 66, possivelmente após uma falha mecânica na suspensão.

Cliff Virginia Tech Grand Challenge Team Virginia Tech., Blacksburg, Va. Parou na milha 44

Rocky Virginia Tech Team Rocky Virginia Tech., Blacksburg, Va. Parou na milha 39 devido a um vazamento no óleo do gerador. Sem gerador os computadores pararam.

ION Desert Buckeyes Ohio State U., Columbus, Ohio Parou na milha 29

DAD Team DAD Digital Auto Drive/Velodyne Acoustics, Morgan Hill, Calif. Parou na milha 26 devido a falha no scanner causada pela vibração.

Desert Rat Insight Racing NC State U., Cary, N.C. Parou na milha 26

Xboxx Mojavaton Grand Junction, Col. Parou na milha 23.5

Golem 2 The Golem Group/UCLA Los Angeles, Calif. Parou na milha 22. Um defeito no software causou uma aceleração a 60mph

CajunBot Team Cajunbot U. of LA, Lafayette, La. Parou na milha 17 quando o atuador do freio pegou fogo após o veículo ter sido pausado por 50 min

RASCAL SciAutonics/Auburn Engineering Thousand Oaks, Calif. Problemas de software: parou na milha 16

Desert Tortoise Intelligent Vehicle Safety Technologies Littleton, Col. Parou na milha 14

NaviGATOR Team CIMAR U. of FL, Gainesville, Florida Parou na milha 14.

Prospect 11 Princeton University Princeton U., N.J. Parou na milha 10

Spider Team Cornell Cornell U., Ithaca, N.Y. Parou na milha 9. Quando foi pausado, bateu em um guard-rail. Quando recebeu a ordem de andar, não conseguir dar ré e se desenvencilhar do gaurd-rail.

Alice Team Caltech Calif. Inst. of Tech., Pasadena, Calif. Parou na milha 8

JackBot MonsterMoto Cedar Park, Texas Parou na milha 7

The Meteor Mitre Meteorites MITRE Corp., McLean, Va. Parou na milha 1

 ©88-09, Pedro Kantek

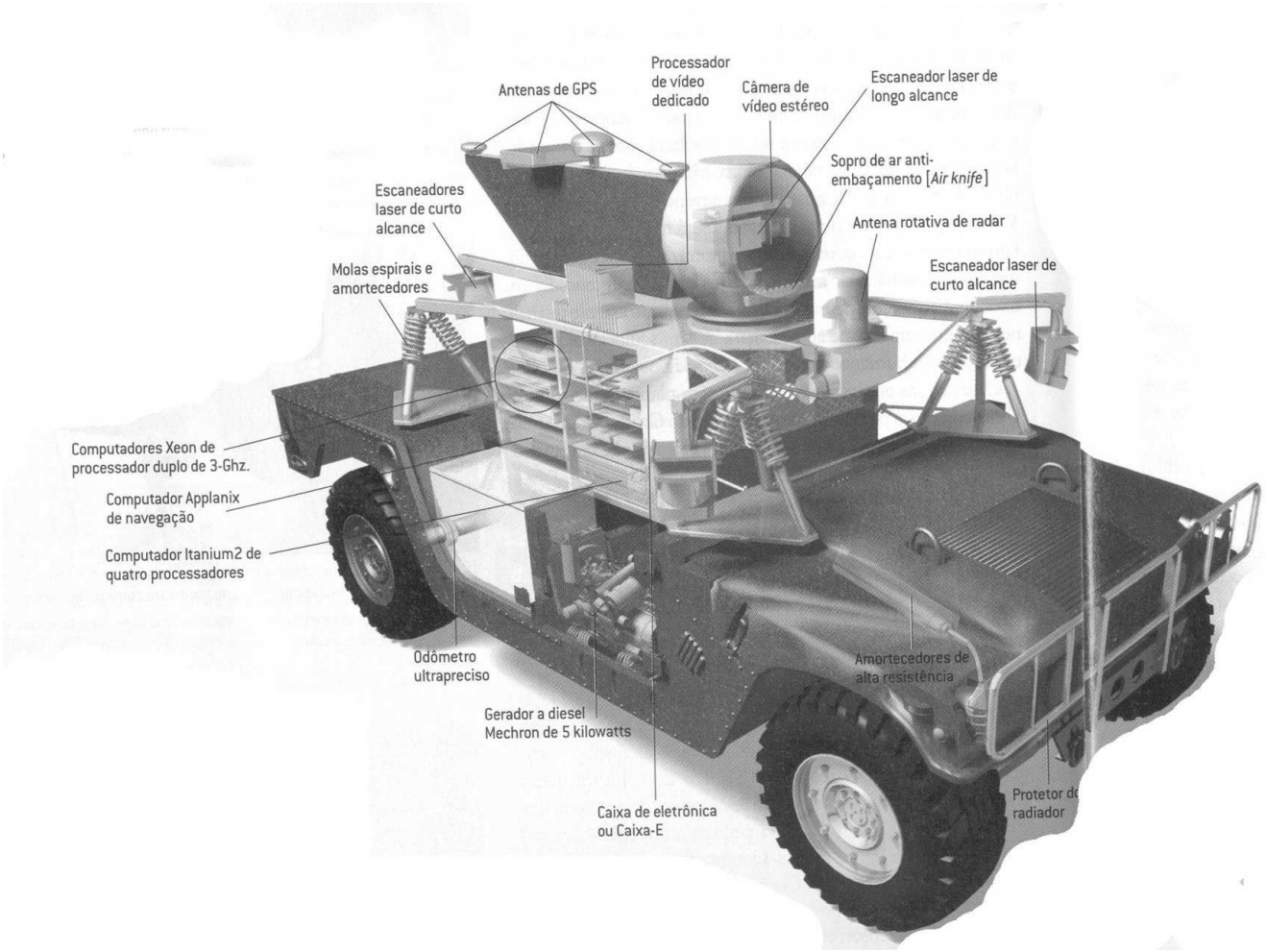
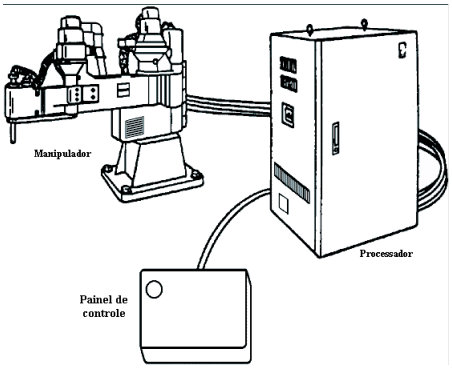


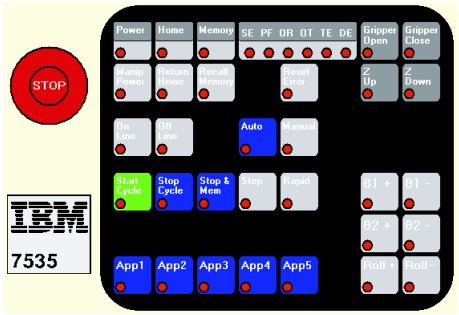
Figura 12.5: A versão 04 do Sandstorm

12.2 IBM7535

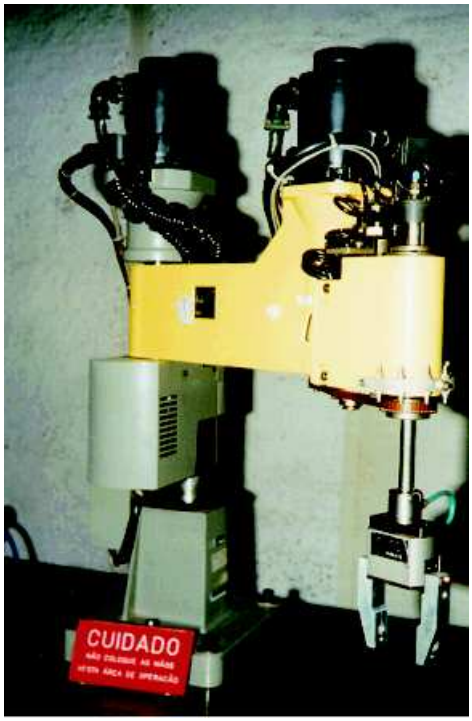
Esta família de robots da IBM está bastante disseminada na indústria. Ela aceita a linguagem de programação AML / Entry versão 3. Eis o aspecto espacial do engenho completo do robot, envolvendo o dito cujo, a central de controle e a área de operação.



A área de interface com o operador (o teclado) apresenta o seguinte aspecto:



Finalmente, uma foto do engenho robótico pronto para operar.

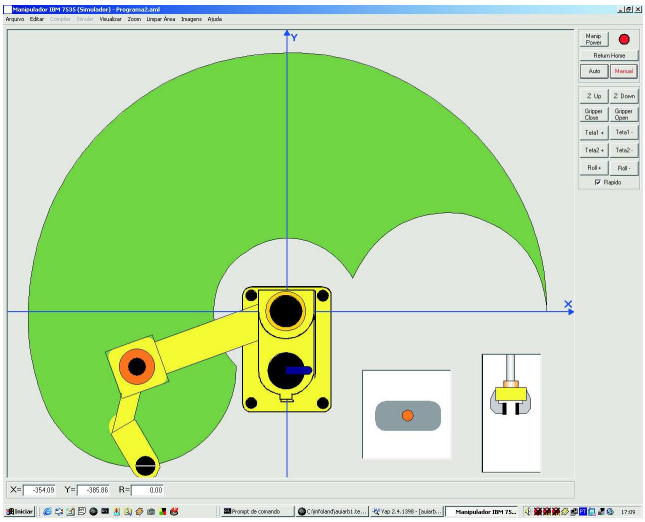


12.2.1 Área de trabalho

O robot manipula uma determinada área que é mapeada em coordenadas X,Y. O eixo central do robot, ocupa a posição 0,0. Quando o robot está na posição inicial, ele se encontra em X=650, Y=0. Quando ocupa a posição mais afastada da inicial, O ponto

 ©88-09, Pedro Kantek

X está em -350 e o Y em -386 (aproximadamente). Eis como fica o espaço de trabalho:



Para familiarizar-se com o funcionamento do robot, utilize o simulador do professor J. Gorgulho, construído na Universidade Federal de Itajubá.

Chame um programa qualquer, e entre na operação manual, operando os controles para ver o jeitão do bruto. Somente depois disso, é que você deve examinar os programas a seguir descritos.

12.2.2 Comandos

São os seguintes os comandos do 7535

Formato	Função	Observação
Branch(label);	Transferir o fluxo do programa para a linha especificada pelo label	Haverá um erro de compilação caso o label não esteja declarado
BreakPoint;	Permite interromper o programa em uma linha específica utilizando o botão "stop and mem" e depois reiniciar deste ponto com os botões "recall memory" e "start"	Um mesmo programa pode conter múltiplos BreakPoints
COLOR(cor)	Define a cor do pincel: 1=preto, 2=vermelho, 3=verde, 4=azul, 5=amarelo, 6=violeta, 7=ciano, 8=laranja, 9=branco	Este comando não faz parte de AML/Entry
nome:STATIC COUNTER;	Define um contador	A faixa válida é de -32.767 a +32.767. O valor inicial é 0. Comandos relacionados: SETC, INCR, DECR, TESTC
DECR(nome);	decrementa o valor do contador citado em 1	Haverá erro se o contador não for definido
DELAY(tempo);	Para o programa pelo intervalo definido. A unidade é 0.1 segundos. O valor permitido varia entre 0.1 e 25.5 segundos	Pode-se usar uma constante como parâmetro
DOWN;	Desce o eixo Z	Normalmente usa-se DELAY depois de DOWN
DPMOVE(dX,dY,dR);	Executa um movimento incremental. dX é a variação de movimento no eixo X, dY no Y e dR a variação de giro no eixo Z	DPMOVE é afetado pela ação do comando LINEAR

 ©88-09, Pedro Kantek

Formato	Função	Observação
END;	Fecha uma rotina iniciada com SUBR	
GETPART(pallet);	Move a garra para a peça atual do pallet citado	Quando o programa é iniciado o pallet não tem peça definida, logo deve-se usar SETPART logo no início. Comandos relacionados: PALLET, NEXTPART, PREVPART, SETPART e TESTP
GRASP;	Fecha a garra	É comum usar DELAY depois deste comando
INCR(nome);	Incrementa o contador em 1	
LINEAR(valor);	Determina o degrau de movimento entre os pontos.	Valor pode ser 0..10, 20, 30, 40 e 50. 0 desativa LINEAR
constante: NEW PT(X,Y,R); constante: NEW n;	Define uma constante que pode ser um ponto ou um valor	
NEXTPART (pallet);	Move a garra para a próxima peça do pallet citado	
pallet: STATIC PALLET(c1, c2, c3, ppl, ntp);	Define um pallet (disposição de peças em grid)	c1: coordenada da primeira peça da primeira fila c2: última peça, primeira fila; c3: última peça, última fila; ppl: peças por linha; ntp: peças totais do pallet
PAYLOAD(valor);	Controla a velocidade e a carga	Valor varia entre 1 e 10. Não é considerado no simulador.
PMOVE(PT(X,Y,R)); ou PMOVE(constante);	Movimenta a garra para as coordenadas indicadas,	Erro haverá se a constante não tiver sido previamente definida
PREVPART(pallet);	Move a garra para a peça prévia do pallet citado.	Quando o programa é iniciado o pallet não tem peça definida, logo deve-se usar SETPART logo no início.
RELEASE;	Abre a garra	É usual usar DELAY após RELEASE
SETC(contador, valor);	Determina o valor inicial para um contador	Faixa válida: -32767 a +32767
SETPART(pallet, posição);	Define a posição atual do pallet	Deve-se usar este comando no início do programa
rotina: SUBR;	Identifica o início de uma sub-rotina	Rotinas não podem conter outras sub-rotinas
TESTC(contador, valor, label);	Compara o contador ao valor. Se igual transfere o programa para o endereço indicado	Tanto o contador quanto o label devem ser previamente definidos.
TESTI(entrada, valor, label);	Compara uma entrada digital. Se igual ao valor do contador desvia para label	
TESTP(pallet, valor, label);	Compara a posição atual do pallet com	

12.2.3 Alguns exemplos

A seguir alguns programas simples deste robot.

Movimento da garra

```
1: – Nome: Programa 1
2: – Função: executar os movimentos da garra
3: – Observação: não há movimento do braço
4: Programa1:Subr;
5: Down;
6: Grasp;
7: Delay(1);
8: Up;
9: Release;
10: Delay(1);
11: End;
```

Diferenças de movimento absoluto e relativo

```
1: – Nome: Programa 2
2: – Função: ilustrar o movimento absoluto (PMove) e incremental (DpMove);
3: Programa2:Subr;
4: PMove(PT(315,490,0));
5: DpMove(-25,70,0);
6: PMove(PT(-620,0,90));
7: DpMove(500,185,0);
8: PMove(PT(650,0,0));
9: End;
```

Uso de contadores

```
1: – Nome: Programa 3
2: – Função: ilustrar o uso de contadores e rótulos;
3: – Observação: acionar a opção Contadores do menu Visualizar.
4: Contador1:Static Counter; – define o primeiro contador
5: Contador2:Static Counter; – define o segundo contador
6: Programa3:Subr;
7: Inicio:; – Rótulo Início
8: TestC(Contador1,10,Final); – Se Contador1=10 salta para Final
9: Incr(Contador1); – Incrementa o Contador 1
10: Branch(Inicio); – Salta para o Início
11: Final:; – Rótulo Final
12: Pmove(Pt(585,205,0)); – Movimenta o braço
13: Pmove(Pt(650,0,0)); – Movimenta o braço
14: Incr(Contador2); – Incrementa o Contador 2
15: SetC(Contador1,0); – Zera o Contador 1
16: Branch(Inicio); – Salta para o Início
17: End;
```

Uso de pallets

```
1: – Nome: Programa 6
2: – Função: demonstrar os uso do Pallet;
```

 ©88-09, Pedro Kantek

```
3: – Observação: Visualize o traço superior (Menu Visualizar).
4: C1:New Pt(100,400,0);
5: C2:New Pt(190,400,0);
6: C3:New Pt(190,310,0);
7: Ppl:New 4;
8: Ntp:New 16;
9: Caixa:Static Pallet(C1,C2,C3,Ppl,Ntp);
10: Programa6:Subr;
11: SetPart(Caixa,1); – Apontador = 1
12: Inicio:;
13: GetPart(Caixa); – Move para a peça do pallet
14: Down; – Desce com a garra aberta
15: Grasp; – Pega a peça
16: Up; – Sobre a garra com a peça
17: Pmove(Pt(-200,550,0)); – Move a peça para a esteira
18: Down; – Desce a peça na esteira
19: Release; – Solta a peça
20: Up; – Sobre com a garra aberta
21: TestP(Caixa,Ntp,Aguarda); – Verifica se é a última peça
22: NextPart(Caixa); – Aponta para a próxima peça
23: Branch(Inicio); – Salta para o início
24: Aguarda:;
25: Pmove(Pt(650,0,0)); – Move o braço para longe
26: TestI(0,0,Aguarda); – Testa a entrada digital (se for 0 salta para Aguarda)
27: End;
```

12.2.4 Para treinar

Escreva dois programas para este simulador. No primeiro, ele deve receber (via uma esteira que vai parar na posição x=300, y=300) um prisma de 3 cubos. Cada um deles com 10cm de aresta. O superior de cor azul deve ser retirado e levado para a posição x=-400, y=400. O segundo, de cor amarela, deve ser levado para a posição x=0, y=500. O terceiro, de cor preta deve ser deixado onde chegou.

Se quiser, reconstrua o prisma em ordem inversa (azul, sobre este o amarelo e sobre este o preto) na posição x=-400, y=400.

O segundo programa deve desenhar um tiro ao alvo formado por 4 círculos de cores diferentes. O diâmetro total do alvo deve ser de 400 unidades. Cada fatia terá portanto uma largura de 50. Disponha a posição central do alvo onde quiser.

Se achar muito difícil (não é) ou muito chato (é) trabalhar com a circunferência, faça um alvo quadrado.

Capítulo 13

Processamento em Linguagem Natural

Talvez a suprema diferença entre um ser humano e um animal irracional seja a linguagem. Não foi por outra razão que Turing, ao elaborar o seu teste de inteligência, usou como ferramenta operacional uma linguagem natural. O uso da linguagem começa com o *ato da fala*. Este termo é aqui usado com o sentido de produzir linguagem (não necessariamente pela boca). Assim, estão incluídas as produções de surdo-mudos usando LABRE, sinais de fumaça feitos por aviões publicitários, mensagens de socorro postas dentro de garrafas e enviadas ao mar, etc. Como não se tem em português uma palavra neutra para disso falar, usar-se-ão as palavras falante, ouvinte e fala nesse sentido mais amplo de emitir e receber comunicação via linguagem.

Uma linguagem vai ser definida como um conjunto (eventualmente infinito) de *cadeias*. Cada cadeia será uma concatenação de *símbolos terminais*, às vezes chamados de *palavras*.

Exemplos

- Na lógica de primeira ordem, os símbolos terminais são P, $\wedge$ e $\neg$. Uma cadeia típica pode ser $P \wedge Q$. A linguagem pode ser a junção de todas as cadeias possíveis.
- Em C, os símbolos terminais são $main$, $printf$, a. Uma cadeia típica pode ser $printf(\text{"}Ovalorprocurado\text{"}, valor)$;
- Em espanhol, os símbolos terminais são yo, el, pan, ... Uma cadeia típica pode ser $unaricaydeliciosatortilla$

Exercícios Escreva os terminais, as cadeias e a linguagem para os seguintes casos:

1. A linguagem dos surfistas
2. A linguagem do Garfield
3. COBOL

13.0.5 Formalismos gramaticais

Chomsky (1957) descreveu quatro classes de formalismos gramaticais que diferem apenas na forma das regras de reescrita. As classes são organizadas como uma hierarquia. Aqui está ela, da classe mais poderosa para a menos.

Recursivamente enumeráveis Utilizam regras irrestritas. Ambos os lados da regra de reescrita podem ter qualquer número de símbolos de terminais e não terminais. Por exemplo, a regra $A\ B \Rightarrow C$. Esta gramática equivale a uma máquina de Turing em seu poder se expressão.

Sensíveis ao contexto O lado direito deve conter pelo menos tantos símbolos quanto o lado esquerdo. O nome *sensível ao contexto* nos diz que numa regra como $A\ S\ B \Rightarrow A\ X\ B$ nos diz que um S pode ser reescrito como um X no contexto de um A antes e um B depois.

Livres de contexto O lado esquerdo consiste em um único símbolo não terminal. Cada regra define a reescrita de um não terminal como lado direito em qualquer contexto. Esta classe é usada para definir linguagens naturais e de programação, embora algumas linguagens naturais tenham construções que não são livres de contexto.

Regulares Toda regra tem um símbolo não terminal do lado esquerdo e um símbolo de terminal, opcionalmente seguido por um símbolo não terminal do lado direito. Esta classe tem poder equivalente ao das máquinas de estados finitos.

Para entender melhor o significado dessas classes, considere as frases "comer uma banana"e "comer uma bandana". Como linguagens livres de contexto, a análise de ambas considerará apenas a diferença entre "banana"e "bandana". Em linguagens sensíveis ao contexto, a análise de "comer o que"ajudará na análise.

As gramáticas situadas mais altas na hierarquia têm maior poder de expressão, mas os algoritmos são mais difíceis e menos eficientes.

Note que as linguagens artificiais formais tem definição matemática rígida. Já as linguagens naturais como espanhol, português, inglês e francês não tem definição rígida, mas são faladas e entendidas por grandes comunidades. No que segue, as linguagens naturais serão tratadas como tendo uma definição mais ou menos rígida.

Uma gramática é um conjunto de regras que especificam uma linguagem. As linguagens formais tem gramáticas finitas e perfeitamente estabelecidas. Já as linguagens naturais são objetos de pesquisa de estudiosos (os lingüistas) que buscam encontrar exatamente a gramática, embora nunca sejam completamente bem sucedidos.

Cada linguagem associa uma certa semântica a cada cadeia válida. Em linguagens naturais surje ainda o conceito de pragmática de uma cadeia, que vem a ser o significado real de uma dada cadeia quando dita em uma situação real. eja por exemplo a cadeia "Que bonito!". Sua semântica é conhecida. Entretanto, há uma diferença entre isto ser dito diante de um espetáculo da natureza (diante das cataratas do Iguaçu, por exemplo), ou diante de uma cena de bagunça: os dois gatos estavam brincando e acabaram de derrubar e quebrar um jarro de cristal. Nestes exemplos temos uma semântica e duas pragmáticas distintas.

A maioria dos formalismos de regras gramaticais se baseia na idéia de estrutura sintagmática: aqui, as cadeias são compostas de subcadeias chamados sintagmas que existem em diferentes categorias. Por exemplo, na frase *o gato caiu da árvore* existem dois sintagmas, o sintagma nominal (o gato) e o sintagma verbal (caiu da árvore). Há duas razões para trabalhar com sintagmas: primeiro eles correspondem a elementos semânticos naturais a partir dos quais os significados podem ser construídos. Por exemplo, os sintagmas nominais se referem a objetos do mundo. Em segundo, categorizar os sintagmas nos ajuda a descrever as cadeias permitidas na linguagem.

Um episódio típico de comunicação, no qual o falante F quer informar o ouvinte O sobre a proposição P usando as palavras M, compreende pelo menos 7 passos:

Intenção F decide querer comunicar P a O.

Geração F planeja como transformar P em uma expressão vocal de modo que seja boa a probabilidade de O ao ouví-la, consiga deduzir P, ou algo similar a P. O resultado deste passo é a geração de M.

Síntese F materializa M. Isso pode ser vibrando o ar, escrevendo na parede, mandando um e-mail, lançando sinais de fumaça. De alguma maneira esta produção deve chegar a O, caso contrário não há fenômeno de comunicação. Chamemos a esta materialização de M'.

Percepção O recebe o fluxo informático como uma realização física na forma de M_2' e a decodifica como as palavras M_2.

Análise O deduz que M_2 tem significados possíveis $P_1, ..., P_n$. Para tanto, O divide sua tarefa em 3 processos:

Análise sintática Busca construir uma árvore de análise para a cadeia de entrada M_2. Os nós interiores da árvore representam *sintagmas* e os nós folha representam palavras.

Interpretação semântica Constrói um significado extraído da expressão analisada em alguma linguagem de representação. Quando a interpretação gerada não é única, diz-se haver ambigüidade.

Interpretação pragmática Busca enriquecer a análise anterior apontando, para o contexto, qual a interpretação mais provável.

Eliminação de ambigüidade O deduz que F pretendia transmitir P_i (onde, no caso ideal $P_i = P$). Aqui O trabalha concluindo qual o significado que o falante pretendia transmitir.

Incorporação O decide acreditar em P_i (ou não).

Existem inúmeras iniciativas que podem estar incluídas em PLN. Por exemplo, um tradutor de idiomas, um consultador de BDs, um robot que atenda ordens humanas. Note-se que um PLN nunca é um fim em si mesmo, exceto para pesquisa. O PLN serve para que o computador entenda coisas escritas em linguagem humana. Seu trabalho é extrair informação da sentença.

Existem inúmeras abordagens para PLN. Esta aula, buscará mostrar 3 abordagens diferentes para o mesmo problema. Eles encontram desempenhos distintos (e conseqüentemente exigem investimentos também distintos) na realização de sua tarefa. Cada um deles, impões também cargas distintas de restrições sobre a síntaxe aceita para funcionar.

Deve-se notar que o objetivo final de uma PLN é descobrir o papel desempenhado por cada uma das palavras ue constituem a frase. De posse desse papel e do significado de cada palavra, que provavelmente esteja dicionarizado, o processador deve ser capaz de inferir a semântica da mensagem.

13.0.6 Máquina de estados

No primeiro, denominado "Máquina de estados", usar-se-á uma gramática bastante restrita na construção das frases. Neste exemplo, as frases serão apenas declarativas (não interrogativas nem negativas) e sempre terão o padrão do português: sujeito, verbo, objeto
Nossa gramática vai ainda:

- Todos os adjetivos precedem o nome que modificam;
- Todos os advérbios sucedem o verbo que modificam;

 ©88-09, Pedro Kantek

- Todas as frases terminam por um ponto.

Exemplos válidos

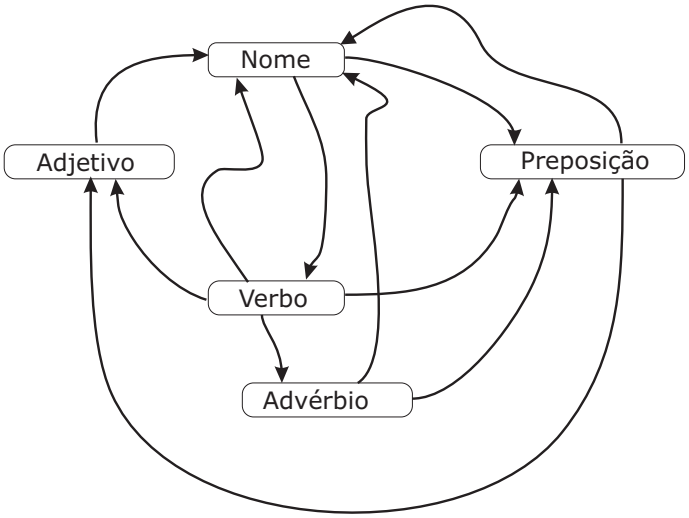
A pedra caiu no chão O veloz time ganhou rapidamente o jogo

Contra-exemplos: O rime veloz ganhou o jogo O veloz time rapidamente ganhou o jogo

Para exemplificar esta abordagem usar-se-á a gramática acima (chamada G_1) e criar-se-á um vocabulário mínimo formado pelas palavras: casa (substantivo), janela (substantivo), porta (substantivo), criança (substantivo), gato (substantivo), cachorro (substantivo), urubu (substantivo), voa (verbo), corre (verbo), mia (verbo), late (verbo), correu (verbo), miou (verbo), voou (verbo), anda (verbo), andou (verbo), muito (advérbio), pouco (advérbio), rapidamente (advérbio), lentamente (advérbio), o (artigo), a(artigo), um (artigo), uma (artigo), para (preposição), grande (adjetivo), pequeno (adjetivo), largo (adjetivo), estreito (adjetivo), bonito (adjetivo), feio (adjetivo) ...

Questões: Note que existe uma preposição (a) e um artigo (a) Note que o tempo (passado, presente, futuro) da ação pode ser inferido pelo tempo do verbo

Neste analisador, uma máquina de estados usa o estado atual da sentença para prever que tipo de palavra pode legalmente vir a seguir. A figura a seguir, mostra a máquina que usaremos a seguir



Trata-se de um grafo dirigido que mostra as transições válidas de um estado a outro. Por exemplo, um substantivo pode ser seguido por um verbo ou por uma preposição. No grafo não aparecem os artigos (que devem ser ignorados), nem o ponto terminador, que encerra a análise.

A função precisa ter acesso a um banco de dados simples, onde estão listadas palavras e sua categoria que pode ser: substantivo, verbo, advérbio, artigo, preposição e adjetivo. Deve ter também uma variável global, que contenha o estado atual da máquina.

Devido ao caráter pedagógico desta função, a mesma apenas assinalará se a frase recebeida está ou não correta. Assim, a função receberá uma frase e devolverá .T. se a mesma estiver correta e .F. senão. Os tratamentos de cada um dos componentes da frase não serão mostrados mas devem estar colocados dentro desta função.

Analisando o desempenho da função na análise da frase *O menino correu rapidamente para a grande casa*. Começa-se pela palavra *O*. Sendo um artigo, o mesmo é descartado e nada acontece. A próxima palavra é *menino* que transforma o estado atual da máquina em SUBSTANTIVO, e com isso a máquina de estados está iniciada.

A próxima palavra é *correu*. Como a figura acima mostrou, existem 2 trasições válidas para substantivo, para um verbo e para uma preposição. Como *correu* é um

verbo, a máquina vai para o estado VERBO. A próxima palavra é *rapidamente* que é um advérbio, e leva a máquina para este estado (ADVÉRBIO). Daqui dois caminhos são válidos: para SUBSTANTIVO e para PREPOSIÇÃO. Como a próxima palavra é *para*, a máquina vai para PREPOSIÇÃO. A próxima palavra é *a* que por ser artigo é desprezada. A seguir, vem o adjetivo *grande* que leva a máquina para este estado (ADJETIVO). A seguir vem o substantivo *casa* que leva a máquina para o estado SUBSTANTIVO. Para terminar, o analisador processa o ponto final e a frase foi toda analisada.

A grande dificuldade desta abordagem é com a complexidade de uma lingua natural (por exemplo, o português). Imagine-se quantos estados existem para uma lingua real. Este fato inviabiliza esta abordagem exceto em casos onde uma gramática restrita (e conseqüentemente artificial) pode ser construída. Outra deficiência importante é que o analisador "não sabe"como chegou a um estado particular. Esta estratégia é indicada em sub-mundos específicos, por exemplo em consultas a bancos de dados.

13.0.7 Analisador Recursivo Descendente

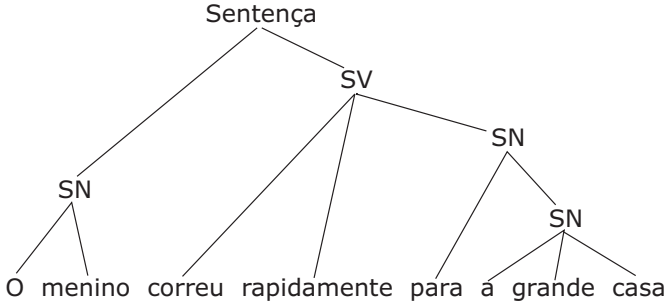
Esta é uma abordagem distinta da anterior. Enquanto aquela era "botton up", esta é "top-down". Aqui a frase candidata é analisada em seu todo, buscando-se (de maneira recursiva, daí o nome) seus componentes. A busca prossegue até que todos os átomos da frase tenham sido detectados. As regras que governam como cada um dos componentes da frase devem ser construídos são chamadas Regras de Produção da Gramática. Esta abordagem é a mais genérica e a que recebe maiores estudos. É também a mais complexa e difícil.

Um analisador livre de contexto usa essas regras para analisar a frase. Eis as regras para a gramática G_1, vista acima:

- Sentença ⇒ SintagmaNominal + SintagmaVerbal
- SintagmaNominal ⇒ artigo + nome
- SintagmaNominal ⇒ artigo + adjetivo + nome
- SintagmaNominal ⇒ preposição + SintagmaNominal
- SintagmaVerbal ⇒ verbo + SintagmaNominal
- SintagmaVerbal ⇒ verbo + advérbio + SintagmaNominal
- SintagmaVerbal ⇒ verbo + advérbio
- SintagmaVerbal ⇒ verbo

O sintagma nominal é recursivo quando uma preposição é encontrada e o sintagma verbal é indiretamente recursivo porque pode invocar um sintagma nominal.

Usando a mesma sentença como exemplo, eis a árvore de análise.



 ©88-09, Pedro Kantek

A análise livre de contexto é usada em quase todas as linguagens de programação (PASCAL, C, BASIC...). Embora seja claramente limitada em sua análise do idioma português (por exemplo), esta técnica tem um subproduto interessante: ela afirma que o português não é apenas um amontoado de dicas e exceções (como custosamente aprendemos na escola), mas é também uma regra absolutamente geral, que todos os falantes e escreventes devem seguir, ainda que nem sempre de maneira consciente (e correta). Outra vantagem desta abordagem é que não apenas átomos podem ser analisados, mas partes de frases e frases completas. Isso pode ter implicações importantes na análise da semântica. Ou seja, agora não apenas detectamos palavras, como também orações.

A análise sintática pode ser definida como o processo de encontrar uma árvore de análise para uma dada cadeia de entrada. Ela pode ser vista como um processo de *busca* em uma árvore de análise. A análise top-down pode ser assim definida:

Estado inicial A sentença S contendo filhos desconhecidos.

Função sucessora Seleciona o nó mais à esquerda com filhos desconhecidos. Depois, busca na gramática as regras que tem o rótulo raiz do nodo no lado esquerdo. Para cada regra, ela cria um sucessor.

Teste de fim Verifica se as folhas da árvore correspondem exatamente à cadeia de entrada, sem incógnitas e sem entradas não incluídas.

O grande problema nesta estratégia é lidar com as regras recursivas à esquerda (por exemplo $X \Rightarrow X$. Esta regra pode levar a uma geração infinita de filhos.

Já a formulação bottom-up da busca é

Estado inicial Cada palavra na entrada é visualizada como uma árvore de análise que tem apenas uma folha.

Função sucessor examina cada posição i na lista de árvores e cada lado direito de uma regra na gramática. Se a lista de árvores casa com o lado direito da regra, a lista de árvores é substituída por uma nova árvore, cuja categoria é o lado esquerdo da regra e cujos filhos são a lista de árvores original.

Teste de fim Achar uma sentença S como raiz de uma árvore.

Note-se que ainda que tivéssemos uma heurística perfeita (que não temos), esses algoritmos ainda seriam ineficientes, já que muitas sentenças tem um número exponencialmente grandes de árvores de análise.

Eis o funcionamento agora: O analisador é chamado e ele achará a oração substantiva seguida da oração verbal e seguida pelo ponto (isso, assumindo que a frase de entrada estará correta). A seguir, cada oração será analisada pela função específica, até a conclusão da verificação da sentença.

Usando este analisador com *O menino correu rapidamente para a grande casa*, é chamado primeiro o analisador de oração substantiva, que percorre a frase e detecta "O menino". Depois o restante é passado ao analisador de oração verbal que detecta um verbo seguido por um advérbio e seguido por uma oração substantiva. O analisador desta é chamado e trata "para a grande casa".

A deficiência desta abordagem é a incapacidade de reconhecer as muitas maneiras válidas de uma frase ser construída em português. A tentativa de implementar tais regras em número crescente levará certamente a uma explosão combinatória. Seja como for ninguém tentou isto até hoje.

Deficiências da gramática acima

- Não trata concordâncias de gênero
- Não trata concordâncias de grau
- Não trata os tempos verbais e a concordância temporal
- Não aborda todas as outras classes gramaticais, que lembrando, são 10 (artigo, verbo, substantivo, adjetivo, advérbio, preposição, conjunção, interjeição, pronome e numeral)
- ambigüidade de palavras: (burro é substantivo ou adjetivo ?)
- Verbos transitivos (direto e indireto), bitransitivos, intransitivos
- Ordem inversa de frases (ouviram do ipiranga...)
- Perguntas: o problema aqui é a chamada "dependência a longa distância- partes de uma frase se referem a partes de outra frase.
- Figuras de linguagem, de pensamento...
- ...o português... (estamos no desvio do "esperanto").
- e o alemão ? *Lebensvericherungsgesellschaftsangestellter* significa funcionário de uma empresa que vende seguros de vida.

Questão: será que a criança já traz uma gramática interna e apenas ajusta parâmetros a respeito dela a partir de exemplos e correções dadas pela mãe, ou será que a gramática é integralmente construída pela criança (ajudada pela professora de português ?)

Semântica Analisada gramaticalmente a frase, deve-se associar um significado a cada sintagma. Como fazer isso, depende da linguagem de representação do conhecimento empregada. Se for a lógica de primeira ordem, então a análise semântica terminará com o assinalamento de uma expressão lógica verdadeira a cada sintagma. Aqui, a grande dificuldade do problema, que é passar de um sintagma a uma expressão lógica de maneira correta e sem ambigüidades. Alguns autores sugerem a utilização de uma forma intermediária de representação para mediar entre a síntaxe e a semântica. Ela pode ser chamada de *forma quase lógica* e ser uma lógica mais relaxada, mais próxima à síntaxe.

EXERCÍCIO 37 Procure compreender o texto, memorizando-o se possível.

> O procedimento é realmente simples. Primeiro você organiza os itens em diferentes grupos. É claro que uma pilha pode ser suficiente, dependendo de quanto existe para fazer. Se tiver de ir para um outro lugar devido à falta de recursos, esta será a próxima etapa; do contrário você estará muito bem adaptado. É importante não se exceder. Ou seja, é melhor fazer poucas coisas de cada vez do que muitas. Em curto prazo, isso pode não parecer importante, mas podem surgir complicações facilmente. Um equívoco também é dispendioso. A princípio o procedimento inteiro parecerá complicado. Porém, logo ele se tornará apenas outro fato da vida. É difícil prever qualquer propósito para a necessidade desta tarefa no futuro imediato, mas nunca se sabe quando vai ser preciso realizá-la. Depois que o procedimento for concluído, o material será organizado de novo em diferentes grupos. Em seguida, eles poderão ser colocados em seus lugares apropriados. Eventualmente, os itens serão usados mais uma vez e o ciclo inteiro terá de ser repetido. Contudo, isso faz parte da vida

 ©88-09, Pedro Kantek

Questões: Procure respondê-las sem olhar de novo para o texto.

1. Quais as quatro etapas mencionadas ?
2. Qual etapa foi omitida ?
3. Qual é o material mencionado no texto ?
4. Que tipo de equívoco seria dispendioso ?
5. É melhor fazer muito ou pouco ? Por quê ?

13.0.8 Analisador por remoção de ruído

Esta estratégia está indicada quando apenas algumas palavras de uma frase carregam informação. O funcionamento é simples. Todas as palavras desconhecidas (que não constam de um determinado dicionário) são consideradas como ruído e são excluídas da frase antes de começar a análise. É usado tipicamente para fornecer comandos a softwares (ou consultar bancos de dados). Para que funcione adequadamente, este analisador impõe uma ordem fixa e pré-estabelecida de palavras na sentença.

Seja por exemplo, um banco de dados com dados de meteorologia. Imagine que ele aceita comandos como

```
mostre todas as capitais do Brasil com temperatura > 20
imprima todas as capitais
mostre curitiba
mostre uma capital com humidade < 80 (não está chovendo)
```

A questão chave aqui é que as sentenças têm uma regra rígida de formação. Usando o mesmo exemplo acima, as sentenças teria a seguinte estrutura:

verbo [modificador][nome][operador valor]

O verbo deve sempre estar presente e os 4 outros componentes são opcionais, porém se operador estiver presente, o valor é obrigatório. Durante a análise toda palavra que não estiver em um dicionário será desconsiderada. A ordem deve ser seguida rigorosamente.

EXERCÍCIO 38 Qual seria a lista de verbos, modificadores, nomes, operadores e valores para um gerenciador de banco de dados de preços agropecuários ?

verbo	modificador	nome	operador	valor

Exemplo de [Rus96]

O autor Russel, no seu livro Inteligência Artificial, apresenta a seguinte gramática, a que ele chama de ϵ_0.

- Sentença ⇒ SintagmaNominal + SintagmaVerbal
- Sentença ⇒ Sentença Conjunção Sentença
- SintagmaNominal ⇒ Pronome
- SintagmaNominal ⇒ Nome

- SintagmaNominal ⇒ Substantivo
- SintagmaNominal ⇒ Artigo + Substantivo
- SintagmaNominal ⇒ Dígito
- SintagmaNominal ⇒ SintagmaNominal SintagmaPreposicional
- SintagmaNominal ⇒ SintagmaNominal CláusulaRelativa
- SintagmaVerbal ⇒ Verbo
- SintagmaVerbal ⇒ SintagmaVerbal + SintagmaNominal
- SintagmaVerbal ⇒ SintagmaVerbal + Adjetivo
- SintagmaVerbal ⇒ SintagmaVerbal + SintagmaPreposicional
- SintagmaVerbal ⇒ SintagmaVerbal + Advérbio
- SintagmaPreposicional ⇒ Preposição + SintagmaNominal
- CláusulaRelativa ⇒ QUE + SintagmaVerbal

Apesar da riqueza maior de ϵ_0 em relação à gramática que estudamos na aula, o livro comenta que ela apresenta **supergeração** já que gera sentenças que não são gramaticais e **subgeração** já que ela rejeita sentenças válidas no idioma.

Desafio

- Escreva e implemente um processador de linguagem natural para automatizar uma estação de pedidos em uma lanchonete McDonalds.

 ©88-09, Pedro Kantek

Capítulo 14

Aprendizado

Parece razoável afirmar que enquanto programas não forem capazes de aprender coisas, não iremos muito longe na IA. Esta característica parece estar intimamente associada à inteligência. Uma razão da dificuldade de enfrentar o aprendizado é que ele engloba inúmeras atividades relacionadas (decisão, criação, linguagem, registro, experimentação...).

Falando grosso modo, existem 3 modelos de aprendizagem

simbólico Aqui, tudo começa com um conjunto de símbolos que possam representar entidades e relações no universo do domínio do problema. Algoritmos de aprendizado tentam inferir generalizações novas, válidas e úteis que possam ser expressas usando estes símbolos.

conexionista Aqui padrões de ligações entre pequenas unidades de processamento individual são ajustados para representar conhecimento. Começando com uma inicialização aleatória, o processamento de dados de treinamento determinam novos valores de pesos e eventualmente de estrutura. Os modelos conexionistas reconhecem padrões invariantes nos dados e representam estes invariantes em sua própria estrutura. Esta abordagem é uma "cópia" do funcionamento do cérebro animal.

evolutivo Já o modelo evolutivo, trata do aprendizado simulando o aprendizado coletivo das espécies animais. Usa-se algo similar a um meio ambiente para que este faça pressão sobre uma população de candidatos, cada um com seu conhecimento e aprendizado devidamente registrados. Com a passagem das gerações, usando-se recombinação e mutações, o aprendizado aparece.

O aprendizado conexionista vai ser analisado quando se estudarem as redes neurais (aula AUIARN1) e o aprendizado evolutivo no tópico computação evolutiva (aulas AUIACEn). Por hora, este tema vai se concentrar no aprendizado simbólico.

14.0.9 Aprendizado simbólico

> *A mera observação de algo não tem valor algum. Observar se transforma em notar, notar se transforma em pensar, pensar se transforma em estabelecer conexões, de maneira que podemos dizer que todo olhar atento que lançamos sobre o mundo é um ato de teorizar. Contudo, isto deve ser feito conscientemente, com autocrítica, com liberdade e, para usar uma palavra audaciosa, com ironia. GOETHE*

O aprendizado é importante para as aplicações práticas da IA. Feigenbaum e McCorduck identificaram o *gargalo da engenharia do conhecimento* como o maior obstáculo

para o uso em larga escala de sistemas inteligentes. Herbert Simon, em 1983, define o aprendizado como *qualquer mudança num sistema que melhore seu desempenho pela segunda vez que ele repetir a mesma tarefa, ou uma outra tarefa da mesma população.* Note-se que aprender implica em generalizar a partir da experiência. Não é apenas a mesma tarefa que deve ser feita melhor: outra tarefas também devem melhorar.

Como os domínios interessantes tendem a ser gigantescos, uma máquina de aprendizagem examinará apenas uma pequena parte dos exemplos possíveis. A partir de uma experiência limitada o aprendiz deve generalizar corretamente para ocorrências do domínio não vistas anteriormente. Este problema é chamado *indução*e é central para o aprendizado. A seleção dos exemplos deve ser heurística, e os critérios de seleção desta heurística são conhecidos como *vieses indutivos.*

As entradas para o aprendizado podem ser diversos tipos: exemplos, contra-exmplos, explicações em alto nível, analogias etc. O armazenamento para o aprendizado também pode ser de vários tipos: expressões lógicas, frames, redes semânticas, tabelas de decisão, tabelas numéricas ou qualquer outra forma de registro *ad hoc* etc. Finalmente, para navegar nesse conteúdo, as máquinas de aprendizagem precisam ter uma estratégia de busca, possivelmente baseada em uma heurística que guie essa mesma busca.

14.0.10 Busca em espaço de versões

Proposto por Mitchell no final dos anos 70 faz um aprendizado indutivo buscando através de um espaço de conceitos. Esta estratégia, usa expressões lógicas e busca a generalização, usando os seguintes métodos

1. Substituir constantes por variáves: Por exemplo, a expressão `cor(carro,preto)` pode ser generalizada como `cor(X,preto)`.

2. Retirar condições em uma expressão conjuntiva (usando $\wedge$). Por exemplo, `preço(X,alto)` $\wedge$ `tamanho(X,grande)` $\wedge$ `cor(X,preto)` poderia ficar `preço(X,alto)` $\wedge$ `cor(X,preto)`

3. Acrescentar um termo disjuntivo a uma expressão. Por exemplo, `preço(X,alto)` $\wedge$ `tamanho(X,grande)` $\wedge$ `cor(X,preto)` poderia ficar `[preço(X,alto)` $\wedge$ `tamanho(X,grande)` $\wedge$ `cor(X,preto)]` $\vee$ `cor(X,verde)`

4. Substituir uma propriedade por seu ascendente em uma hierarquia de classes. Se soubermos que `corbásica` é uma superclasse de `preto`, então `cor(X,preto)` é generalizado como `cor(X,corbásica)`

A generalização pode ser estudada com base na teoria dos conjuntos. Sejam P e Q os conjuntos de sentenças que casam com as expressões do cálculo de predicados p e q respectivamente. A expressão p é mais geral do que q se e somente se $P \supseteq Q$.

No exemplo acima, o conjunto de sentenças que casa com `cor(X,preto)` contém o conjunto de elementos que casam com `cor(carro,preto)`. Note que a expressão "mais geral que" define uma ordenação parcial no espaço de sentenças lógicas. Expressamos isso usando o símbolo $\geq$, onde $p \geq q$ significa que p é mais geral do que q. Essa ordenação é uma fonte poderosa de restrições na busca realizada por um algoritmo de aprendizagem.

Formalizamos isso através da notação de **cobertura**. Se o conceito p é mais geral do que o conceito q dizemos que p cobre q.

Exemplo

Seja um exemplo, através de um domínio com os seguintes objetos e propriedades, supostamente falando de livros:

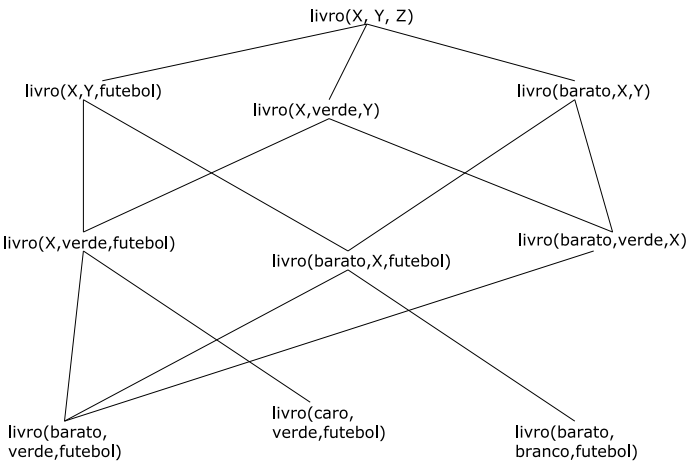


Figura 14.1: um espaço de conceitos

```
preço={barato, caro}
cor={preto, branco, verde}
assunto={futebol, história, matemática}
```

Usando o predicado `livro={preço, cor, assunto}`. A operação de generalização de substituir constantes por variáveis define o espaço da figura 14.1. Podemos ver a aprendizagem indutiva como busca neste espaço por um conceito que seja consistente com todos os exemplos do treinamento

14.0.11 Algoritmo de eliminação de candidatos

Os algoritmos a seguir são devidos a Mitchell,1982. Eles se baseiam na noção de *espaço de versões* que é o conjunto de todas as descrições de conceitos consistentes com os exemplos do treinamento. Os algoritmos atuam reduzindo o tamanho do espaço conforme mais exemplos se tornam disponíveis. São 3 algoritmos: o primeiro atua do específico para o genérico. O segundo, do genérico para o específico e o terceiro vai nas duas direções, encontrando-se no meio. Os 3 algoritmos são guiados por dados. Como eles usam dados de treinamento, diz-se que realizam aprendizagem supervisionada.

Embora seja possível generalizar apenas a partir de exemplos positivos, os negativos são importantes ao evitar que o algoritmo generalize excessivamente. Em outras palavras, o conceito aprendido deve ser suficientemente geral para cobrir todos os exemplos positivos e deve ser suficientemente específico para deixar de fora todos os exemplos negativos.

No espaço da figura 14.1 um conceito que cobriria todos os conjuntos de exemplos exclusivamente positivos seria simplesmente `livro(X,Y,Z)`. Mas, salta aos olhos que este conceito é geral demais porque ele implica que todos os exemplos pertencem ao conceito-alvo. Uma maneira de evitar a supergeneralização é generalizar tão pouco quanto possível para cobrir os exemplos positivos; uma outra forma é usar exemplos negativos para eliminar conceitos gerais superpostos. Como ilustra a figura 14.2 exemplos negativos evitam a supergeneralização forçando a máquina de aprendizagem a especializar conceitos.

Para o conjunto de hipóteses S, define-se a busca do específico para o geral como:

 ©88-09, Pedro Kantek

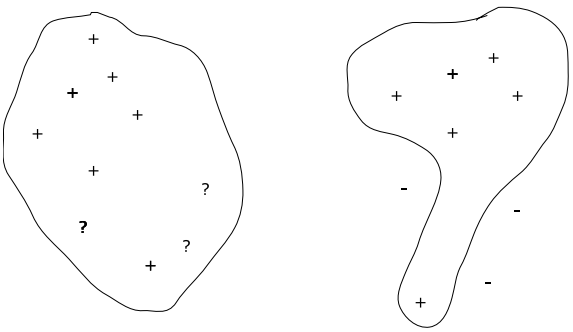


Figura 14.2: exemplos negativos versus generalização

```
1: Coloque em S o primeiro exemplo positivo
2: N é o conjunto de todos os exemplos negativos vistos até agora
3: para exemplo positivo p faça
4:    para s ∈ S faça
5:       se s não casar com p, substitua s pela sua generalização mais específica que case
         com p
6:    fimpara
7:    retire de S todas as hipóteses que são mais gerais do que outra hipótese em S
8:    retire de S as hipóteses que casem com um exemplo de N
9: fimpara
10: para exemplo negativo n faça
11:    retire os membros de S que casem com n
12:    acrescente n a N
13: fimpara
```

A busca do específico para o geral mantém um conjunto S de hipóteses ou definições candidatas de conceitos. Acompanhe a construção de S para o conceito "bola"

1. S:{} e os atributos do objeto são obj(tamanho,cor,tipo)

2. chega um exemplo positivo: obj(pequeno,vermelho,bola), e o resultado de S é {obj(pequeno,vermelho,bola)}

3. chega o exemplo positivo: obj(pequeno,branco,bola), e o resultado de S é {obj(pequeno,X,bola)}

4. chega o exemplo positivo: obj(grande,azul,bola), e o resultado de S é {obj(Y,X,bola)}

14.1 Qual a idade dos meus 3 filhos ?

Dois amigos que não se viam a muitos anos se encontram na rua. Conversam sobre diversos assuntos e lá pelas tantas, um deles diz "como sei que você é um professor de matemática, aqui está um problema para você resolver. Hoje é um dia especial para mim. Todos os meus 3 filhos fazem aniversário hoje. Você poderia me dizer quais as idades de cada um ?"

"Claro", responde o matemático, "mas você terá que me dar algumas informações adicionais".

"Está certo" responde o amigo, "você terá algumas informações adicionais: a idade deles multiplicada é igual a 36".

"Muito bem" diz o matemático, "mas eu ainda preciso mais".

"A soma das suas idades é igual ao número de janelas naquele prédio" diz, apontando para a casa da esquina.

O matemático ainda gasta algum tempo pensando e replica "ainda preciso de mais informação".

"Meu filho mais velho tem olhos azuis" diz o amigo. "Então, é o suficiente" responde o matemático, e em seguida informa ao amigo as 3 idades dos filhos. E, é claro, os números estão corretos.

Como foi que o matemático resolveu o problema ?

Começando pelo primeiro pedaço de informação. Se o produto das 3 idades é 36, existem apenas 8 casos a considerar, já que as idades são inteiras (o aniversário é hoje !)

idade do filho 1	do filho 2	do filho 3
36	1	1
18	2	1
12	3	1
9	4	1
9	2	2
6	6	1
6	3	2
4	3	3

O segundo pedaço de informação, informa que a soma das idades é igual ao número de janelas de um prédio. Imaginando que este número seja conhecido pelo matemático, vamos somar os 3 números nas 8 possibilidades

36	+	1	+	1	=	38
18	+	2	+	1	=	21
12	+	3	+	1	=	16
9	+	4	+	1	=	14
9	+	2	+	2	=	13
6	+	6	+	1	=	13
6	+	3	+	2	=	11
4	+	3	+	3	=	10

Se o número fosse 38 (ou 21, 16, 14, 11 ou 10), o problema estaria resolvido. Como o matemático pediu mais dados, só sobrararm as opções 9, 2 e 2 ou 6, 6 e 1. Finalmente, quando o amigo disse que "O" filho mais velho tinha olhos azuis, o matemático matou a charada.

As idades dos 3 filhos são 9, 2 e 2 anos.

14.2 Alguns conceitos caros à IA

14.2.1 A volta de Santa Catarina

Suponha que foi criada uma corrida de bicicletas no Estado de Santa Catarina, ao estilo do "Tour de France". São diversas etapas, dando a volta ao estado, e em cada etapa os vencedores vão acumulando pontos. A chegada é em Floripa e em geral, apenas nessa etapa é que é conhecido o campeão da Volta de Santa Catarina.

 ©88-09, Pedro Kantek

Em um dia qualquer, um jornalista começa a escrever

> Hoje, grande etapa: Contorno de Joinville, com 11 voltas do contorno da cidade em 5800 m por volta. Temos 6 corredores na liderança da corrida. São eles...

Desafortunadamente, muito emocionado pelo desenrolar da prova, nosso jornalista se atrapalha e confunde corredores, seus números, a marca pela qual correm e o estado de cada um. Vamos ajudá-lo a redigir uma notícia correta ?

Nós sabemos que

- O grupo é composto de 6 ciclistas, um de cada estado diferente: um alagoano, baiano, carioca, gaucho, sergipano e paranaense.
- Três empresas patrocinam os corredores, cada empresa a dois deles.
  1. O corredor número 1 e o alagoano são patrocinados pelas Gasosas Cini
  2. O corredor número 5 e o carioca levam as cores do mercado Angeloni
  3. O gaucho e o número 3 representam a Caloi
- Os corredores 2 e 6 saíram na frente do circuito, na entrada da rodovia Dona Francisca, enquanto que o gaúcho ficou para trás.
- O sergipano e o paranaense se adiantaram 30 segundos sobre o número 3 na terceira volta deste circuito de 5800 m.
- O número 2 e o alagoano precisaram abandonar a prova depois de uma queda.
- Finalmente, o número 1 ganhou a corrida, na frente do sergipano.

A pergunta que se faz é quem são os corredores, quais seus números e que empresa cada um deles representa.

14.2.2 Tentativa de solução

Antes de prosseguir no exercício, tente resolver o problema proposto. As explicações metodológicas que seguem serão tão mais produtivas se já houver intimidade com o problema. Apenas para registro, uma pessoa normal deve levar entre 15 minutos e 2 horas para resolvê-lo, usando um ou mais métodos dos a seguir propostos.

14.2.3 Diversos métodos de solução

A seguir diversos enfoques para solução deste problema (e porcerto, de outros parecidos).

1. Método cronológico

A análise se faz segundo a cronologia do texto, ou seja, na ordem em que as informações são apresentadas. A coisa começa com 3 informações:

- Cini: corredor 1 e o alagoano
- Angeloni: corredor 5 e o carioca
- Caloi: corredor 3 e o gaucho

Depois, vamos para a estrada Dona Francisca. Aqui sabemos que o 2 e o 6 adiantaram-se em relação ao gaúcho. Como acima listaram-se 3 números, (1, 5 e 3) e aqui referenciam-se mais dois (2 e 6), logo a conclusão é que o gaúcho é o número 4.

A seguir, na terceira volta, por idêntica análise, descobre-se que o baiano é o número 3.

Na queda, o 2 e o alemão abandonam. Daqui se conclui que o alagoano é o 6 e que o carioca é o 2.

Na chegada, o número 1 ganha do sergipano. Conclui-se que o sergipano é o número 5 e que o paranaense é o número 1. Veja na figura 1, o resultado destas elocubrações:

lugar	dedução	informação
Joinville	Cini:1+alagoano angeloni:5+carioca caloi:3+gaúcho	
Dona Francisca	2 e 6 na frente do gaúcho	4 é o gaúcho
3^o volta	sergipano e paranaense na frente do 3	baiano é o 3
queda	alagoano e o 2 caem	alagoano é o 5 e o carioca é o 2
chegada	o 1 e o sergipano chegam	o sergipano é o 5 e o paranaense é o 1

2. Organizando os dados

Nem sempre a cronologia pode ser usada, nem sempre os dados estão organizados e arrumados. Então, pode ser necessário depurar melhor as informações e neste caso, vamos construir um quadro do que é possível e do que é impossível. Para ajudar na escrita, vamos identificar cada informação

a O 1 e o alagoano são patrocinados pelas Gasosas Cini

b O 5 e o carioca levam as cores do mercado Angeloni

c O gaucho e o número 3 representam a Caloi

d 2 e 6 saíram na frente do circuito, na entrada da rodovia Dona Francisca, o gaúcho ficou para trás.

e O sergipano e o paranaense se adiantaram 30 segundos sobre o número 3 na terceira volta.

f O número 2 e o alagoano abandonaram depois de uma queda.

g O 1 ganhou a corrida, na frente do sergipano.

Neste caso, como o objetivo é atribuir um número a cada corredor, pode-se concluir que

 ©88-09, Pedro Kantek

O 1 **não** pode ser alagoano (a) carioca (marcas diferentes) gaúcho (marcas diferentes) sergipano (g)	Logo, pode ser o paranaense ou o baiano
O 2 não pode ser gaúcho (d) alagoano (f)	pode ser baiano carioca sergipano paranaense
O 3 não pode ser o gaucho (c) carioca (marcas diferentes) alagoano (marcas diferentes) sergipano (e) paranaense (e)	pode ser o baiano
O 4	pode ser baiano, carioca, paranaense, sergipano, alagoano ou gaúcho
O 5 não pode ser carioca (f) alagoano (marcas diferentes) gaúcho (marcas diferentes)	pode ser baiano, paranaense ou sergipano
O 6 não pode ser o gaúcho	pode ser o baiano, alagoano, sergipano, paranaense ou carioca

A solução agora é obtida analisando a coluna número 2 do quadro acima. Note-se que a identificação dos corredores varia entre um (para o número 3) até seis para o número 4. Identificando a partir das possibilidades em ordem crescente, o número 3 é o baiano, o número 1 é o paranaense, o número 5 é o sergipano, o número 2 é o carioca, o número 6 é o alagoano e o número 4 é o gaúcho.

3. Usando um quadro de dupla entrada

Um quadro de dupla entrada permite cruzar duas variáveis e assim fazer o repertório dos diferentes casos possíveis. No nosso enigma existem 3 variáveis (marca, naturalidade e número). Para usar este esquema privilegiaremos a naturalidade e o número, deixando a marca como atributo secundário. Na figura 2, o quadro após preenchido com as informações das 3 primeiras afirmações.

n./est	baia	alag	cari	gaúc	serg	para	
1	o		o	o			Cini (a)
2							
3	o		o	o			Caloi (c)
4							
5	o		o	o			Angeloni(b)
6							

Foi colocada na tabela acima as informações procedentes das afirmações (a), (b) e (c), e também o que pôde obter por dedução. Assim, o corredor 1 não pode ser o alagoano (o que se codifica colocando um "o"na casa correspondente), não pode ser o carioca (posto que este é patrocinado pelo Angeloni (b), e a linha 1 é da gasosa cini (a), e não pode ser o gaúcho, pois este é patrocinado pela Caloi (c) e a linha 1 é da Cini (a). A mesma coisa para os atletas 3 e 5. Ao final, tem-se um quadro com nove "o"que traduzem as impossibilidades do problema.

A seguir, preenche-se o quadro com as demais informações, o que leva à figura 3.

n./est	baia	alag	cari	gaúc	serg	para	
1	o		o	o	o(g)		Cini (a)
2	o(f)			o(d)			
3	o		o	o	o(e)	o(e)	Caloi (c)
4							
5	o		o	o			Angeloni(b)
6					o(d)		

As conclusões que permitem resolver o problema são as mesmas das estratégias anteriores e não serão repetidas aqui.

4. Integrama: diagramas com 3 variáveis

Embora seja uma invenção recente, este diagrama já começa a aparecer em obras de referência. Apenas serão esboçados os passos necessários para usá-lo. Na resolução são tratados simultaneamente 3 quadros de dupla entrada Assim, vemos 2 corredores da Gasosa Cini, identificados por Ci_1 e Ci_2, os corredores do Angeloni (An_1 e An_2)

	1	2	3	4	5	6	Ci_1	Ci_2	An_1	An_2	Ca_1	Ca_2
alag	o						o	1a	o	o	o	o
baia								o		o		
cari					o		o	o	o	1b	o	o
gauc								o		o		
serg								o		o		
para								o		o		
Ci_1	1a	o	o	o	o	o						
Ci_2	o				o							
An_1	o	o	o	o	1b	o						
An_2	o				o							
Ca_1	o				o							
Ca_2	o				o							

A seguir, coloca-se a informação dos corredores da marca Caloi (Ca_1 e Ca_2)

 ©88-09, Pedro Kantek

	1	2	3	4	5	6	Ci_1	Ci_2	An_1	An_2	Ca_1	Ca_2
alag	o						o	1a	o	o	o	o
baia								o		o		
cari					o		o	o	o	1b	o	o
gauc			o				o	o	o	o	o	1c
serg								o		o		
para								o		o		
Ci_1	1a	o	o	o	o	o						
Ci_2	o		o		o							
An_1	o	o	o	o	1b	o						
An_2	o		o		o							
Ca_1	o	o	1c	o	o	o						
Ca_2	o		o		o							

O leitor poderá continuar o preenchimento do integrama. Note-se que este diagrama é mais ou menos universal e pode ser aplicado a qualquer problema que tenha natureza similar ao aqui estudado.

5. Diagrama adaptado à natureza do problema

As informações (a), (b) e (c) podem ser representadas em um esquema onde as variáveis figuram alinhadas

número	1		5		3	
natur		alag		cario		gauc

Sendo o primeiro grupo da empresa Cini (condição a), a segunda do Angeloni (b) e a terceira da caloi (c).

Vemos ai aparecerem dois grupos, o primeiro é o dos corredores de número ímpar que são o paranaense, o sergipano e o baiano. O segundo é o dos corredores de número par que são o alagoano, o carioca e o gaúcho.

Pela condição (d) sabemos que o gaúcho não é o 2 nem o 6, e como, segundo vimos acima não é o 1, nem o 5 e nem o 3, logo se conclui que o gaúcho é o número 4. Também, usando a informação (e) se obtém que o baiano é o número 3. Agora sabe-se tudo sobre os corredores da Caloi

3	4
baiano	gaúcho

Por (f), sabemos que o número 2 não é o alagoano, logo ele é o carioca. Assim, o alagoano só pode levar o número 6, que é o único disponível.

Finalmente, por (g), não sendo o número 1 o sergipano, ele é o paranaense, e o sergipano leva o número 5. Daqui, que a solução é:

número	1	6	5	2	3	4
natur	paran	alago	sergi	cario	baian	gauch

6. Resolução usando a linguagem dos conjuntos

Estamos na presença de 6 corredores cada um deles representando um estado e cada um tendo um número. Podem-se descrever o conjunto de estados **{A,B,C,G,S,P}** e o conjunto dos números **{1,2,3,4,5,6}**. Pode-se afirmar que os dois conjuntos representam a mesma coisa (ou seja, são de certa forma, iguais). Escreve-se: textbf{A,B,C,G,S,P} = **{1,2,3,4,5,6}**

A partir dessa igualdade, o objetivo é associar a cada letra um número. Das informações (a), (b) e (c) temos dois igualdades de conjuntos: **{B,S,P} = {1,3,5}** e **{A,C,G} = {2,4,6}**

A informação (d) pode ser escrita como **{2,6,G} = {2,4,6}** pois **G=4** e **{2,6} = {A,C}**.

A informação (e) se escreve **{S,P,3} = {B,S,P}** pois **B=3** e **{S,P} = {1,5}**.

A informação (f) se escreve **{2,A} = {2,6}** pois **A=6** e **C=2**.

A informação (g) se escreve **{1,S} = {S,P}**.

Esta solução é muito próxima à resolução com diagramas vista anteriormente. O diagrama visualiza a estrutura, como é também o caso da formalização em termos de linguagem dos conjuntos: vê-se aparecer o esqueleto do enigma.

7. Dois quadros de dupla entrada

Voltemos à primeira parte da solução proposta em 3. Considerar as informações (a), (b), e (c) nos leva para a figura

n./est	baia	alag	cari	gaúc	serg	para	
1	o		o	o			Cini (a)
2							
3	o		o	o			Caloi (c)
4							
5	o		o	o			Angeloni(b)
6							

Se olharmos este qudro, podemos escrever um primeiro quadro referentes aos números ímpares e que tem associado a igualdade **{B,S,P} = {1,3,5}**. E um segundo quadro referente aos números pares e associando a igualdade **{A,C,G} = {2,4,6}**. Colocando nestes quadros as informações (d), (e), (f) e (g), as respostas saem de imediato.

	B	S	P
1		o(g)	
3		o(e)	o(e)
5			

	A	C	G
2	o(f)		o(d)
4			
6			o(d)

Para encerrar

Alguns leitores podem ficar surpresos de não encontrar a forma que usaram para resolver o enigma aqui nesta lista. Existem muitos métodos de solução, e provavelmente é impossível ser exaustivo nessa apresentação. Foram mostrados aqui instrumentos e métodos bastante típicos: eles traduzem diferentes modos de captar os problemas. Finalmente, é bom lembrar que nada impede o uso de métodos mistos, usando um pouco de cada um.

 ©88-09, Pedro Kantek

14.2.4 Diagramas e esquemas: a árvore

A árvore nos ajuda a esquematizar os dados de entrada em uma classe grande de problemas, conhecidos como combinatórios. Vamos olhar a técnica a partir de um exemplo

Exemplo de árvore

Seja um trecho de contrato bancário

> Os critérios para obtenção de empréstimos são poucos. Existem duas taxas praticadas. A normal, que é de 6% ao mês e a taxa preferencial, que é de 5,5 % a.m. Clientes antigos, que não tenham restrições de crédito, fazem juz a taxa preferencial. Clientes novos, sem restrições, fazem juz a taxa normal. Clientes que tenham restrições, se forem antigos, e tiverem rendimentos acima de R$ 2000,00/mensais pagarão taxa normal. Todos os demais pedidos serão rejeitados.

Examinando os dados, percebe-se a existência de 3 critérios que são importantes para a concessão ou não do crédito. São eles

- A faixa de renda do cliente (com limite de R$ 2000)
- A restrição ao crédito
- A antigüidade do cliente

Organizando a árvore, escolhe-se um dos critérios (qualquer um) para separar os filhos da raiz. Neste exemplo, vai-se escolher a faixa de renda do cliente. Fica:

1. clientes com menos de R$ 2000 de renda
2. clientes com R$ 2000 de renda ou mais

Depois, inclui-se a restrição ao crédito e fica

1. clientes com menos de R$ 2000 de renda
   (a) clientes com restrição
   (b) clientes sem restrição
2. clientes com R$ 2000 de renda ou mais
   (a) clientes com restrição
   (b) clientes sem restrição

Finalmente coloca-se a antigüidade do cliente

1. clientes com menos de R$ 2000 de renda
   (a) clientes com restrição
      i. cliente novo
      ii. cliente antigo
   (b) clientes sem restrição
      i. cliente novo
      ii. cliente antigo
2. clientes com R$ 2000 de renda ou mais

   (a) clientes com restrição
      i. cliente novo
      ii. cliente antigo
   (b) clientes sem restrição
      i. cliente novo
      ii. cliente antigo

E agora, colocam-se a permissão e a taxa de emprestimo de cada categoria

1. clientes com menos de R$ 2000 de renda
   (a) clientes com restrição RESULTADO: NÃO
   (b) clientes sem restrição
      i. cliente novo RESULTADO: TX NORMAL
      ii. cliente antigo RESULTADO: TX PREFERENCIAL
2. clientes com R$ 2000 de renda ou mais
   (a) clientes com restrição
      i. cliente novo RESULTADO: NÃO
      ii. cliente antigo RESULTADO: TX NORMAL
   (b) clientes sem restrição
      i. cliente novo RESULTADO: TX NORMAL
      ii. cliente antigo RESULTADO: TX PREFERENCIAL

 ©88-09, Pedro Kantek