Notas de aula

por Pedro Kantek
pkantek@gmail.com ou
pkantek@up.com.br

A estrela

Instituição	Universidade Positivo
Curso	Engenharia da Computação
Disciplinas	
Aula	
Exercícios associados	
Versão deste documento	13 de julho de 2012

Para saber mais

Sumário

1 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 [1]

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.

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.

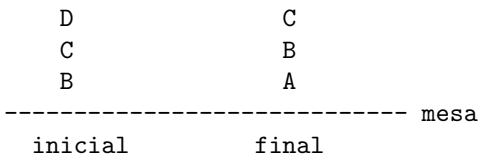
Existem outras heurísticas mais ou menos gerais. Por exemplo, a seguinte de Lenat normalmente ajuda: Se há uma função interessante de dois argumentos $f(x, y)$, analise o que acontece se x for igual a y.

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:

```
A          H
H          G
G          F
F          E
E          D
```

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

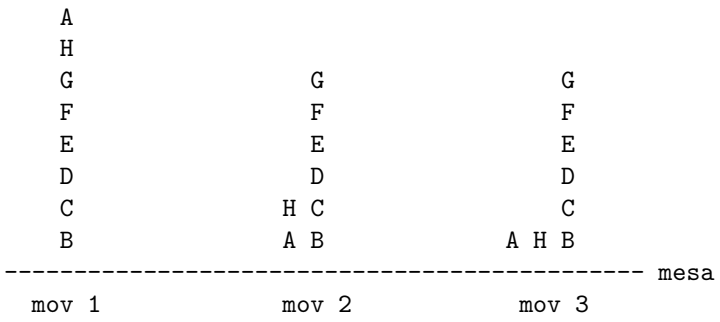


Suponhamos que os operadores são:

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

1.1 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.

1.2 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.

1.3 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). 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.

1.3.1 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: $h_1 =$ _____, $h_2 =$ _____

Problemas com informação parcial

Até agora os ambientes eram observáveis e determinísticos, nos quais o agente sabe qual o efeito de cada ação. Portanto o agente pode calcular para onde vai e onde está. O que acontece quando o conhecimento dos estados ou ações é incompleto ? Há 3 casos a estudar

problemas de conformidade Ou problemas sem sensores. O agente não sabe onde está. Portanto cada ação pode levar a diversos estados sucessores.

problemas de contingência o agente pode perceber novas informações após cada ação. Cada percepção possível define uma contingência que deve ser planejada. Um problema é chamado adverso se a incerteza é causada por outro agente.

problemas de exploração Estados e ações do ambiente são desconhecidos. O agente deve atuar para descobrí-los.

Busca informada

Cap. 4 do Russell.

Busca gulosa

A abordagem geral aqui é a busca pela melhor escolha. Agora o nodo selecionado para expansão é o vencedor em uma função de avaliação $f(n)$. Tradicionalmente o nodo com a menor avaliação é o selecionado, porque a avaliação mede a distância até o objetivo. O nome "busca pela melhor escolha" é inapropriado. Se a melhor escolha realmente estivesse disponível não seria uma busca e sim uma marcha em direção ao objetivo. Escolhe-se o nodo que *parece* ser o melhor de acordo com a função de avaliação. Existem diversos algoritmos aqui, todos com funções de avaliação diferentes. Chamar-se-á essa função de $heuristica(n)$, onde n é um determinado nodo e combina-se que $h(n) = 0$ quando n é o objetivo.

A busca gulosa pela melhor escolha usa uma heurística que escolhe o nodo mais próximo ao objetivo. A interpretação da palavra "próximo" depende do problema que se está a estudar.

No exemplo da Romenia, este algoritmo chega ao resultado com 3 estados, mas o resultado não é ótimo. Outro caminho é 32km mais curto.

A*

É a função mais amplamente usada. Se puder se garantir que h(n) nunca superestimará o custo para atingir o objetivo então pode-se provar que a resposta é ótima. Um exemplo aqui é a distância em linha reta entre duas cidades (nada pode ser menor que a reta entre elas).

Se fizer uma busca em grafo ao invés de busca em árvore, o caráter de otimicidade pode ser derrubado (o segundo caminho é melhor que o primeiro...). Para lidar com isso, duas soluções:

1. Ao achar um duplo, descartar o mais caro

2. Impor a h(n) a consitência ou monotonicidade. Isto significa gerar o mais barato antes dos demais.

O livro não cita, mas é importante falar dos papeis de k_1 e k_2 em

$$Min(C) = k_1 \times p(n) + k_2 \times h'(n)$$

Busca heurística limitada pela memória

O fato de A* precisar manter todos os nodos em memória, levando ao seu esgotamento antes do tempo, levou ao desenvolvimento de outros algoritmos:

A* de aprofundamento iterativo É um A* em que o corte usado é o custo de $f(g + h)$, em vez da profundidade g. A cada iteração o valor de corte é o menor valor de f de qualquer nodo que tenha excedido o corte na interação anterior.

Um melhoramento é a **busca recursiva pelo melhor**. Sua estrutura é semelhante a uma busca recursiva em profundidade. Porém ao invés de descer indefinidamente pelo caminho atual ela controla o valor de f do melhor caminho alternativo disponível a partir de qualquer ancestral do nodo atual. Se nodo atual excede esse valor, ela retorna ao ancestral.

Outro ainda é o **A* limitado pela memória simplificado**. Prossegue como o A* até preencher a memória. Neste ponto para gerar um novo nodo algum precisa ser descartado. E, o escolhido é pior nodo folha, e copia o valor de f no seu pai. Para evitar o empate (descartar uma e voltar a gerá-la: isto ocorre se todos os nodos tiverem o mesmo custo), o algoritmo expande a melhor folha mais nova e descarta a pior folha mais antiga.

Funções heurísticas

O quebra cabeça 8 tem solução em média de 22 passos. Com um fator de ramificação médio de 3, tem $3^{22} = 3.1 \times 10^{10}$ estados. Controlando os estados repetidos baixa-se 170.000 vezes esse número, pois só há 9!/2=181.400 estados. O número correspondente para o quebra-cabeça 15 é próximo a 10^{13}. Daí o texto define as duas heurísticas usuais:

- a quantidade de peças no lugar errado
- a distância Manhattan para as peças irem para o lugar certo.

Uma maneira de caracterizar uma heurística é o fator de ramificação efetiva b^*. Por exemplo, se A* encontra uma solução na profundidade 5, usando 52 nodos, a ramificação efetiva será 1,92. Para testar as funções heurísticas h_1 e h_2, geraram-se 1200 problemas aleatórios com tamanhos de solução variando entre 2 a 24 e resolveram-se os problemas com busca por aprofundamento iterativo (BAI), A*(h1) e A*(h2), originando a seguinte tabela

	custo da busca			ramificação efetiva		
d	BAI	A*h1	A*h2	BAI	A*h1	A*h2
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	6384	39	25	2,80	1,33	1,24
10	47127	93	39	2,79	1,38	1,22
12	3644035	227	73	2,78	1,42	1,24
14		539	113		1,44	1,23
16		1301	211		1,45	1,25
18		3056	363		1,46	1,26
20		7276	676		1,47	1,27
22		18094	1219		1,48	1,28
24		39135	1641		1,48	1,26

Segue-se uma discussão sobre um critério para criar heurísticas boas é considerar o problema relaxado. Trata-se de problema similar com menos restrições. No caso do quebra-cabeça 8, o movimento pode ocorrer se

A pode ir para B se e somente se A e B são horizontal ou verticalmente adjacentes e B é vazio

o relaxamento poderia ser:

- um bloco pode ir de A a B se A é adjacente a B

- um bloco pode ir de A a B se B é vazio
- um bloco pode ir de A a B

Se tivermos diveras heurísticas, pode-se ter o melhor dos mundos fazendo

$$h(n) = max\{h_1(n), ..., h_n(n)\}$$

Usando-se bancos de dados de heurísticas parciais, baixa-se o número de nodos na busca do QC-15 em 10.000 vezes e para o QC-24 diminui-se 1.000.000 vezes.

Outra possibilidade é criar heurísticas que aprendam a partir da experiência. Resolvendo uma grande quantidade de instâncias do QC, cada exemplo consiste num estado do caminho de solução e no custo real da solução a partir desse ponto. Isso pode ser feito por redes neurais, árvores de decisão e outros. Pode-se usar também características (quantos pares já estão certos ? quantos blocos mal posicionados ? e fazer $h(n)$ ser uma combinação linear de tais características.

Há outra classe de problemas (as n rainhas, por exemplo) que não interessam pelo caminho, ou seja, este é irrelevante, e sim apenas pelo estado final. Os algoritmos de busca local operam usando um único estado corrente em vez de vários caminhos e se movem apenas para os vizinhos desse estado.

Problemas de otimização pura usam métodos locais também. Um deles é o método da subida da encosta, também chamado de busca gulosa local. Aparecem problemas, como por exemplo

máximos locais ponto mais alto que todos os seus vizinhos, mas que não é o máximo local.

picos sequencia de picos próximos e regularmente espaçados

platôs área com a função de avaliação plana

Para corrigir isso propuseram-se: subida da encosta estocástica (escolha ao acaso uma das subidas); subida da encosta pela primeira escolha (escolha da primeira real subida depois de várias gerações ao acaso); subida da encosta com reinício aleatório (diversos pontos de partida aleatórios).

Outro método é a têmpera simulada: junta dois algoritmos interessantes (uma subida da encosta que nunca faz movimentos para baixo é eficiente. uma subida que faz movimentos aleatórios acha um bom resultado, mas é ineficiente). Na têmpera simulada no início são admitidos grandes saltos e depois só pequenos saltos. Na hora de escolher a nova direção, uma aleatória é escolhida. Se ela melhorar está aceita. Se ela piorar, será aceita com uma probabilidade menor que 1. A probabilidade diminui à medida em que "o material esfria".

Outro algoritmo é a busca em feixe local. Ele começa com k esttados gerados aleatoriamente. Em cada passo são gerados todos os sucessores dos k estados. Se qualquer um deles for o objetivo o algoritmo pára. Senão, ele selecionará os k melhores e o ciclo continua. Este método acrescenta um grau importante de paralelismo. Basta compará-lo com k execuções aleatórias independentes. Uma maneira de evitar a convergência precoce neste método é aceitar alguns sucessores ruins.

Algoritmos genéticos

bla bla bla

Agentes de busca on line

Um agente on-line é aquele que precisa processar dados à medida em que eles são recebidos em vez de esperar a disponibilidade do conjunto todo. Um agente off-line calcula tudo e depois vai para a ação. Um agente on-line intercala computação e ação. Executa algo e olha para os lados. (um exemplo simples é o xadrez: não dá para calcular a partida toda)

A busca online é necessária em um problema de exploração em que os estados e ações são desconhecidos. O exemplo canônico é um robot colocado em abiente desconhecido. Neste caso não se pode conhecer os sucessores de um estado sem antes ir para ele.

Problema de satisfação de restrições

Extende o conceito de problemas de busca em espaço de estados para problemas de satisfação de restrições, onde usam-se heurísticas de uso geral (ao invés de específicados do problema).

Formalmente, um PSR é definido como um conjunto de variáveis $X_1, X_2, ..., X_n$ e por um conjunto de restrições $C_1, C_2, ..., C_n$. Cada variável X_i tem um domínio D_i não vazio de valores possíveis. Cada

restrição envolve um subconjunto das variáveis e especifica combinações de valores para aquele conjunto. Um estado do problema é definido por uma atribuição de valores a alguma ou a todas as variáveis. Uma solução é uma atribuição a todas as variáveis sem violar alguma restrição. Pode-se ter o requisito adicional de maximização.

É util visualizar um PSR como um grafo de restrições. Os nodos do grafo correspondem a uma variável e os arcos a restrições.

UM PSR também pode receber uma formulação como busca padrão:

estado inicial a atribuição vazia a todas as variáveis

sucessor atribuição de um valor a qualquer variável, desde que não entre em conflito com as variáveis já atribuídas

objetivo atribuição corrente é completa

custo custo constante (por exemplo=1) para todo passo.

A solução deve ser uma atribuição completa, portanto estará na profundidade n, e a árvore só se estende até lá. Por isso, usa-se a busca em profundidade. Além disso o caminho é irrelevante. Portanto, pode-se usar uma formulação completa que pode ou não atender às restrições, e daí usa-se a busca local.

Os casos mais simples de PSR têm variáveis discretos e domínios finitos. Há também PSR booleanos (as soluções podem ser {V,F}. Resolver um problema como PSR ao invés de busca convencional permite tratar instâncias várias ordens de grandeza maior.

Quando as variáveis têm domínios infinitos, não é mais possível enumerar os valores das restrições. Usa-se então uma linguagem de restrições. A categoria mais simples de PSR de domínios contínuos são os da programação linear.

As restrições podem ser unárias (australianos do sul rejeitam a cor verde), binárias (tazmania $\neq$ vitoria), ou de ordem mais alta (problemas criptoaritméticos).

Para estes o livro apresenta a solução da conta

```
  TWO
 +TWO
 ----
 FOUR
```

e apresenta as seguintes equações: $O + O = R + 10.X_1$
$X_1 + W + W = U + 10.X_2$
$X_2 + T + T = O + 10.X_3$
$X_3 = F$

Daí ele desenha um grafo (pag.138 do Russell) que transforma restrições de alta ordem em restrições binárias (duas a duas) e diz que sempre é possível fazer isso.

Daí vem a questão das restrições serem absolutas ou preferenciais. O exemplo do professor que prefere dar aulas pela manhã. Este tipo de restrição é tratada fazendo-se custos crescentes para as opções menos desejadas.

Ao resolver um problema de criptoaritmética usando a busca em largura, tem-se que o fator de ramificação no nivel superior é nd porque qualquer valor dos d pode ser atribuído a qualquer das n variáveis. No próximo nivel o fator é $(n-1) \times d$ e assim por diante até n niveis. A árvore terá $n! \times d^n$ folhas embora existam apenas d^n atribuições possíveis.

Para lidar com este fato, relembra-se que os PSR tem a propriedade da comutatividade (isto é ordem de aplicação não tem nenhum efeito sobre o resultado. Portanto, todos os algoritmos de busca em PSR geram sucessores considerando atribuições possíveis para apenas uma única variável em cada nó da árvore de busca.

Usa-se a **busca com retrocesso** neste caso e o retrocesso ocorre quando uma variável não tem valores válidos para atribuição. Para fazer isso é importante estabelecer uma certa ordem de atribuição dos valores e uma boa heurística aqui é ordenar as variáveis por ordem decrescente de valores a serem atribuídos a ela. Isto caracteriza a heurística **VRM-valores restantes mínimos** que garante ótimos resultados.

O VRM não ajuda nada na hora de escolher a primeira variável. Nessa hora usa-se a **heurística de grau** que sugere pegar primeiro a variável com mais restrições.

Escolhida uma variável, é hora de atribuir-lhe um valor . A heurística do **valor menos restritivo** sugere usar o valor que elimina o menor número possível de variáveis a seguir.

Pode-se fazer a **verificação prévia**. Sempre que uma variável X é atribuída examina-se cada variável Y ainda não atribuída e excluí-se do domínio de Y qualquer valor que seja incompatível com a atribuição de X.

Uma generalização deste processo é a **propagação de restrições** que sugere levar a verificação prévia mais à frente. O algoritmo aqui é a **consistência de arco** sobre o grafo de restrições.

Sempre que uma verificação prévia baseada nas atribuições a X exclui um valor do domínio de Y, ela deve adicionar X a um *conjunto de conflito de Y*. Este conjunto é importante, porque ao fazer o retrocesso no processo de busca, este retrocesso não precisa ser unitário (deixa o algoritmo lento) e sim deve ser feito até o último elemento incluido no conjunto conflito.

Busca local para PSR

Aqui começa-se com um estado e ele vai sendo modificado até satisfazer o objetivo. A escolha de um novo valor para uma variável, a heurística mais óbvia é selecionar o valor que resulta no número mínimo de conflitos com outras variáveis, chamada **heurística de conflitos mínimos**. Esta heurística resolve eficientemente o problema de um milhão de rainhas e o do programa de observação do Hubble de 3 semanas para 10 minutos.

Uma boa maneira de tratar este tipo de problema é buscar reduzir o grafo de restrições a uma árvore. Fazendo isso a solução ocorrerá em tempo linear em relação ao número de variáveis.

1.4 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

1.5 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:

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

1.6 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

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

1.7 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:

1.7.1 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*.

1.7.2 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.

1.7.3 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.

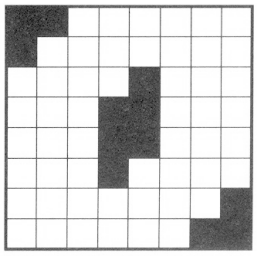
Um bom exemplo desta técnica está em http://www.sbmac.org.br/tema/seletas/docs/v5_2/00-Souza.pdf

1.7.4 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



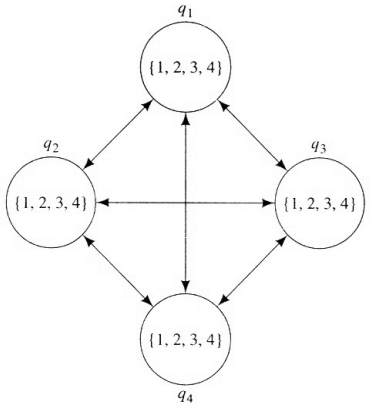
O objetivo é rechear a grade com letras de maneira que as linhas e as colunas formem palavras válidas em um determinado idioma. Este problema, se tratado como busca convencional de estados, tem um espaço imenso. Cada espaço na grade pode recebe 1 de 26 letras. Um estado intermediário poderia ser uma 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.

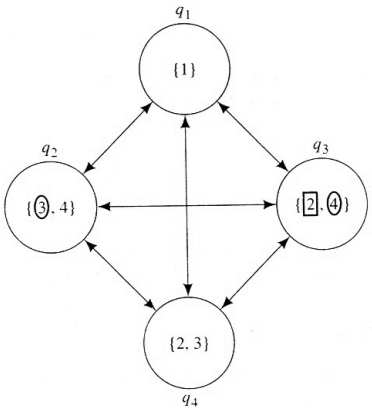
Todos os arcos da figura acima são arco-consistentes, já que para cada par de nodos q_i e q_j (onde $i \neq j$) e para cada valor de q_i existe um valor de q_j que não viola as restrições.

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 $q_4 = 4$.

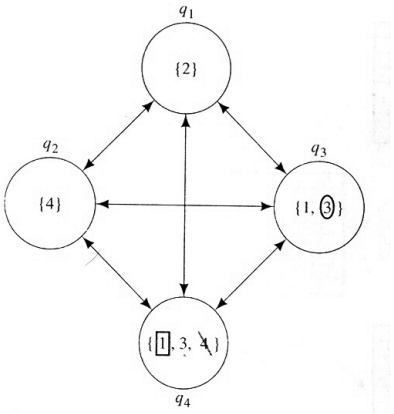
Obtem-se o grafo seguinte



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 $q_1 = 2$.



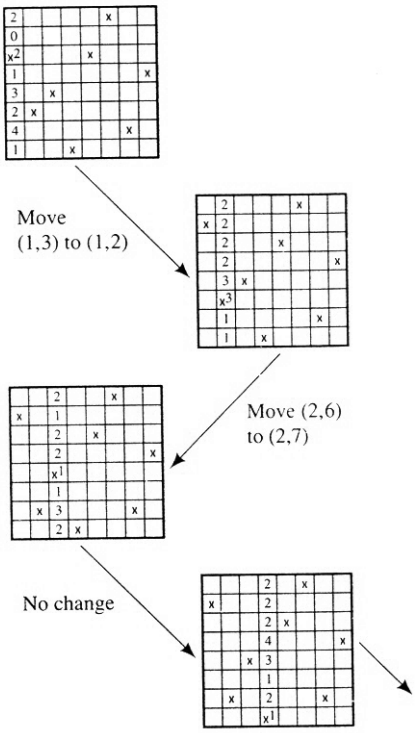
Acabamos de achar uma solução, sem ser necessário pesquisar todos os estados no espaço de estados.

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.

1.7.5 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.