Problema do caixeiro viajante - TSP

O problema do caixeiro-viajante (PCV), ou como é conhecido em inglês: travelling salesman problem (TSP), é um problema de otimização que, apesar de parecer modesto é, na realidade, um dos mais estudados, por cientistas, matemáticos e investigadores de logística, genética e produção, entre outros (Applegate et al., cop. 2006, p. 1). O problema pertence à categoria NP-hard (em inglês), que o remete para um campo de complexidade exponencial, isto é, o esforço computacional necessário para a sua resolução cresce exponencialmente com o tamanho do problema. Assim, dado que é difícil, se não impossível, determinar a solução ótima desta classe de problemas, os métodos de resolução passam pelas heurísticas e afins que, do ponto de vista matemático, não asseguram a obtenção de uma solução ótima (Cunha, 2002).

Definição e formulação do problema Em termos leigos, imaginem um território de vendas (por exemplo, o Estado do Paraná), que tem diversas cidades polo que devem ser visitadas pelo vendedor. Ele começa e termina na sua base e deve visitar uma e só uma vez cada cidade polo. O critério para a escolha da ordem das cidades é minimizar a distância percorrida.

Usando uma definição acadêmica:

> O problema do caixeiro-viajante consiste na procura de um circuito que possua a menor distância, começando numa qualquer cidade, entre várias, visitando cada cidade precisamente uma vez e regressando à cidade inicial (Nilsson, 1982).

O tamanho do espaço de procura aumenta exponencialmente dependendo da quantidade de cidades n, o número de cidades, uma vez que existem $(n-1)!/2$ circuitos possíveis (a posição inicial é arbitrária, e a ordem do circuito pode ser invertida).

A solução do PCV pode ser de dois tipos: a exata (impraticável em instância médias ou grandes, devido ao comportamento exponencial) e as aproximadas, todas dependentes de alguma heurística.

Começando pelo segundo grupo, há diversos métodos. Estes são procedimentos particulares, o que os torna inflexíveis para a determinação de boas soluções para um outro problema ligeiramente diferente.

Quadro esquemático de soluções aproximadas

- construção de circuítos (alg. guloso)
- árvore de cobertura mínima
- *simulated annealing*
- algoritmo de busca tabu
- redes neurais
- algoritmos genéticos
- *aint colony optimization algorithm*

Solução Exata Embora demorada esta é a solução que vai ser trabalhada neste exercício. É um bom exemplo de algoritmo recursivo (neste caso, claramente a recursão vem em nosso auxílio: este mesmo algoritmo não recursivo teria enorme complicação e dificuldade) e suficiente pequeno para poder rodar em minutos usando um computador pessoal simples.

Um caso real Em 1998 uma equipe de matemáticos encontrou o caminho mais curto para visitar as 13.509 cidades americanas que tinham, naquele ano, mais de 500 habitantes. Foram necessários 3,5 meses de processamento de três multiprocessadores (32 pentium cada) ligados em rede. Aqui, eliminaram-se as rotas obviamente ineficientes logo de cara. O problema: a estratégia só vale para este problema e para estas cidades.

Outro caso real Os algoritmos mostrados nesta folha e rodados em um micro bem lerdinho (um pentium 4 com 2.8GHz, com 632 MB de RAM, mas rodando Linux e neste sob WMware um Windows XP, e neste rodando APL2 da IBM) e em um micro rápido (2 CPUS de 2.80GHz, 3.3GB de memória e com APL2 sob Windows XP nativo)

qtd de cidades	CPU lerda	CPU rápida
6 cidades	31 milisseg	desprezível
7 cidades	156 milisseg	78 milisseg
8 cidades	1.1 seg	500 milisseg
9 cidades	9 seg	4 seg
10 cidades	81 seg	37 seg
11 cidades	836 seg	367 seg
12 cidades	153 min	67 min
13 cidades	30.6 horas	13.4 horas
14 cidades	16.5 dias	7.2 dias

O problema que você vai resolver

Há uma lista de 10 cidades paranaenses, cada uma com suas coordenadas obtidas a partir de um eixo cartesiano imaginário montado sobre o paralelo 26^o SUL (x) e sobre o meridiano 54^oO (y). No cálculo, considerou-se uma projeção plana do território, o que certamente introduzirá pequenas distorções. Não importa, avante ! Seja um exemplo de 10 cidades:

```
 1-CAMPO MOURAO        164 216
 2-CURITIBA            467  63
 3-FCO BELTRAO          91  -8
 4-GUAIRA              -25 213
 5-IRATI               338  62
 6-LONDRINA            291 297
 7-ORTIGUEIRA          310 202
 8-PALMAS              200 -54
 9-PARANAGUA           543  51
10-PARANAVAI           158 323
```

Supondo que a cidade inicial seja 2 (Curitiba), a sequência de cidades a visitar é 2 5 8 3 4 1 10 6 7 9 2 e a distância é de 1560.3 km

Algoritmos Para construir a matriz de distâncias entre pares de cidades, o algoritmo é

```
 1: funcao ACHADIST (COORD: nome,xx,yy[nn]) : matd [nn;nn]
 2: i,j : inteiro
 3: para i = 1 to nn
 4:     para j = 1 to nn
 5:         matd[i][j] ← COORD[i;2 3] distancia COORD[j;2 3]
 6:     fim{para}
 7: fim{para}
```

Note-se que a função `distancia` acima usada é a simples aplicação do Teorema de Pitágoras sobre o mapa. Note-se também que se o TSP deve ser feito sobre outro conceito que não a distância linear, esta função deve ser substituida pela que calcula o método desejado. Este pode ser: distância via estradas, tempo de voo, custo do pedágio, ...

Montada a matriz de distâncias, é hora de chamar a função TSP, mas antes, são necessárias algumas variáveis globais: MENOR: a menor distância até agora encontrada e inicializada com $+\infty$ e CMENOR: o caminho que dá origem à menor distância até agora encontrada.

```
 1: funcao TSP (matd, dateaqui,futuro,passado)
 2: se futuro[1]≠ 1
 3:     J ← 1
 4:     enquanto J ≤ futuro[1]
 5:         D1 ← dateaqui +
 6:         matd[passado[1+passado[1]][futuro[1+J]]
 7:         novofuturo ← excluivv (futuro[J+1], futuro)
 8:         novopassado ← passado
 9:         novopassado[1] ← novopassado[1]+1
10:         novopassado[1+novopassado[1]]← futuro[J+1]
11:         TSP (matd,D1,novopassado,novofuturo)
12:         J ← J + 1
13:     fim{enquanto}
14: senão
15:     D1 ← dateaqui +
16:     matd[passado[1+passado[1]][futuro[2]]
17:     D1 ← D1+matd[futuro[2]][passado[2]]
18:     passado[passado[1]+2]←futuro[2]
19:     passado[1]←passado[1]+2
20:     se D1 < MENOR
21:         MENOR ← D1
22:         CMENOR ← passado
23:     fim{se}
24: fim{se}
```

Os vetores `passado` e `futuro` ambos tem na primeira posição um contador de valores válidos (para usar parâmetros de tamanho fixo), e eles contém as cidades ja visitadas (passado) e as cidades a visitar (futuro). A função `excluivv` subtrai um no contador do vetor e exclui no vetor a cidade citada no parâmetro.

TSP e o algoritmo guloso Se uma resposta degradada for aceitável, pode-se aplicar o algoritmo guloso neste problema. A estratégia aqui é *das cidades não visitadas, devo escolher a mais próxima.* Em um caso real, obtiveram-se os seguintes valores:
Cidades: 1-Cascavel, 2-Fco Beltrao, 3-Guaira, 4-Guarapuava, 5-Irati, 6-Ortigueira, 7-Paranavai, 8-Ponta Grossa, 9-Telêmaco Borba e 10-Umuarama. Inicial: Ponta Grossa:
Solução exata: 1111.97 Km. Sequência: 8 5 4 2 1 3 10 7 6 9 8. Algoritmo Guloso: 1329.56 Km. Sequência: 8 5 4 6 9 7 10 3 1 2 8

☞ Para você fazer

Suponha a seguinte lista de cidades

```
 1-ADRIANOPOLIS        495 146
 2-CURITIBA            467  63
 3-FOZ IGUACU          -59  55
 4-GUARAPUAVA          254  67
 5-IRATI               338  62
 6-LARANJ SUL          159  68
 7-PALMAS              200 -54
 8-PARANAGUA           543  51
 9-PARANAVAI           158 323
10-TELEM BORBA         341 185
```

Sendo que o início e o fim do ciclo de viagens deve ocorrer na cidade de PALMAS . Após calcular o menor caminho entre todas as cidades, responda:

1. Qual o comprimento em quilômetros do menor caminho ?
2. Qual o NOME da primeira/última cidade a ser visitada ?
3. Qual o NOME da última/primeira cidade a ser visitada ?

distância	prim/ult	ult/prim