ECT2303 – Linguagem de Programação

LAB 10 – Vamos aplicar nosso conhecimento em programação a um problema NP-difícil?

1 Introdução

Você decidiu que irá conhecer as 10 maiores cidades do Rio Grande do Norte em termos de número de habitantes. Estas cidades estão dispostas na Tabela 1. As distâncias entre estas cidades são apresentadas no ANEXO I, ao final deste documento.

Tabela 1: Cidades mais populosas do Rio Grande do Norte

Código	Cidade	Habitantes
1	Natal	869 954
2	Mossoró	288 162
3	Parnamirim	242 384
4	São Gonçalo do Amarante	98 260
5	Macaíba	78 021
6	Ceará-Mirim	72 878
7	Caicó	71 238
8	Assu	57 292
9	Currais Novos	44 887
10	São José de Mipibu	43 191

PERGUNTA:

> *Partindo-se de Natal, qual a ordem de visitação das cidades, sem passar mais de uma vez pela mesma cidade e voltando ao ponto de partida, cujo deslocamento em Km é mínimo?*

Confuso? É provável. Veja a Tabela 2 para o número de soluções possíveis para este problema em razão do número de cidades. O problema apresentado, na literatura, é conhecido como o Problema do Caixeiro Viajante (PCV), um dos problemas mais clássicos da área de otimização combinatória.

Tabela 2: Número de cidades *versus* número de soluções

Número de Cidades	Número de soluções
10	181,440
15	4.4E+10
20	6.1E+16
25	3.1E+23
50	3.0E+62
100	4.7E+155

RESPOSTA:

> ⇒ *Natal* ⇒ *Parnamirim* ⇒ *São José de Mipibu* ⇒ *Macaíba* ⇒ *Currais Novos* ⇒ *Caicó* ⇒ *Mossoró* ⇒ *Assu* ⇒ *Ceará-Mirim* ⇒ *São Gonçalo do Amarante* ⇒ *Natal*

Para a rota dada, considerando-se as distâncias definidas no ANEXO I, a distância total a ser percorrida no passeio é de 814.5 Km. A solução apresentada é a **solução ótima** do problema

considerado e 814.5 é seu **custo**. Dado que a solução mostrada é a solução **ótima**, sabe-se, matematicamente, que não existe outra solução para este problema cujo custo seja menor. Pode haver, no entanto, outra solução de mesmo custo.

O problema PCV é classificado como NP-difícil. Isto significa que não se conhece um algoritmo de tempo polinomial que o resolva de forma **ótima** para qualquer instância existente. Em outras palavras, pode-se dizer que, dependendo do número de cidades, é inviável se obter a resposta ótima em um tempo aceitável. Este processo poderia, por exemplo, levar alguns séculos.

Um exemplo de algoritmo que resolve este problema de forma **exata** (ótima) é o algoritmo de *branch-and-bound*. Este algoritmo foi utilizado para resolver o exemplo mostrado, haja vista que o número de cidades é relativamente pequeno. Utilizou-se para tal o *software* CPLEX, que possui uma implementação deste algoritmo e pode ser utilizado por meio de diversas linguagens de programação, como por exemplo, C, C++, Java e Phyton, além de um ambiente gráfico com linguagem própria (OPL). O CPLEX, ou resolvedor, levou apenas 0.04 segundos para obter a resposta para nosso exemplo.

Linguagens de modelagem como GAMS e AMPL também são suportadas. Para utilizar estas ferramentas, é necessário que se tenha um conhecimento mínimo de programação, assim como conhecimento matemático para formular um problema de otimização. O CPLEX é fornecido gratuitamente para fins de pesquisa e ensino. Caso queira estudar mais sobre este *software*, pergunte ao professor (ou então ao Google).

Mas, como dito anteriormente, o PCV é NP-difícil, e isto quer dizer que nem sempre algoritmos exatos serão capazes de resolvê-lo em um tempo aceitável para determinado conjunto de dados.

Isto mesmo! Nem a ferramenta da IBM dá conta.

Sabendo disto, que nem sempre é possível se obter a resposta ótima para um problema em um tempo aceitável, muitas vezes os tomadores de decisão se contentam em obter apenas uma solução de boa qualidade para o problema. Esta solução não necessariamente é a ótima. Isto é comum em áreas como Engenharia.

Para obter estas soluções de boa qualidade, pesquisadores e profissionais do mercado recorrem a uma família de métodos chamados de heurísticas. Uma definição possível para um método heurístico é a seguinte: uma heurística é um algoritmo que fornece uma solução para um problema. Note que esta definição é muito ampla. Sendo assim, qualquer ideia que gere uma solução válida para o problema pode ser classificada como uma heurística.

Sua vez

Sabendo o que é uma heurística, tente imaginar alguma que forneça uma solução para o problema apresentado. Gaste ao menos 5 minutos pensando em um modo de resolver o problema antes de prosseguir.

Se você pensou em, partindo de Natal, visitar sempre a cidade mais próxima a cidade em que atualmente está, parabéns. Você pensou em uma Heurística Gulosa, uma abordagem muito conhecida na literatura.

Se você pensou em qualquer outra forma de resolver o problema, parabéns também. Você pensou também em uma heurística. Lembre-se, uma heurística é um algoritmo que fornece uma solução para um problema.

Mas o que você deve saber é que, nem sempre uma heurística fornece uma solução de boa qualidade. Na verdade, a solução fornecida pode estar muito distante da solução ótima. Considerando esta realidade, os pesquisadores conceberam uma família complementar de métodos, chamados de metaheurísticas.

Metaheurísticas são métodos que preveem a aplicação de estratégias de busca mais especializadas às heurísticas, permitindo que estas encontrem soluções de boa qualidade para o problema em questão. A partir do uso de metaheurísticas, ainda, é provável que a solução ótima de um problema seja encontrada. Porém, uma metaheurística não fornece a prova matemática de que a solução dada é a solução ótima, isto é, a melhor solução possível. Os métodos exatos, diferentemente, fornecem esta prova.

A vantagem em se utilizar metaheurísticas advém do fato de que na maioria das vezes a ideia por trás destes métodos é simples, assim como sua implementação. Há também uma enorme economia de tempo e recursos computacionais ao se utilizar estes métodos. Há, desta forma, a possibilidade de se obter soluções de boa qualidade para problemas complexos em muito menos tempo e com recursos computacionais muito mais limitados em relação aos métodos exatos.

Na literatura, são encontradas diversas heurísticas e metaheurísticas, como por exemplo:

- Colônia de formigas;
- Têmpera Simulada;
- Busca em Vizinhança Variada;
- Busca Tabu;
- Otimização por Nuvem de Partículas;
- Algoritmos Genéticos;
- Outros;

Muitas das ideias por trás da forma como estes métodos encontram soluções para os problemas de otimização são baseadas em observações de fenômenos físicos ou naturais. Por exemplo, Algoritmos Genéticos se baseiam na Teoria da Evolução das Espécies, especialmente o mecanismo de seleção natural. Colônia de Formigas, por sua vez, é baseada na forma como as formigas encontram alimento. Já algoritmos baseados em Têmpera Simulada, são baseados no processo de resfriamento de metais. Caso tenha ficado curioso, faça uma pesquisa sobre estes métodos.

Outra metaheurística conhecida na literatura é a **Busca Local Iterada (BLI)**, mais conhecida do inglês *Iterated Local Search - ILS*. O Capítulo a seguir descreverá esta abordagem, a qual utilizaremos para resolver o PCV.

2 Busca Local Iterada

Para descrever o algoritmo BLI são necessários poucos elementos. Porém, para cada problema a ser resolvido, estes elementos devem ser adaptados. Isto é, a ideia por trás do método é genérica, porém seus elementos precisam ser customizados para o problema que se quer tratar. Veja o Algoritmo 1, no qual a BLI é apresentada em sua forma mais genérica. Para o algoritmo apresentado, temos as seguintes subrotinas:

1. *gera_solução_inicial*: obter uma solução inicial para o problema. Para este passo, qualquer heurística que gere uma solução válida pode ser usada;
2. *busca_local*: consiste em, a partir de uma solução existente, obter uma solução de melhor custo utilizando alguma estratégia de movimentação;
3. *perturbação*: a partir de uma solução existente, o processo de perturbação deverá gerar uma nova solução. Preferencialmente, este processo deve ser feito com base em movimentos aleatórios.

A BLI basicamente repete estes três procedimentos enquanto algum critério de parada não for satisfeito. Este critério, por exemplo, pode ser o tempo total de execução do algoritmo ou o número de iterações.

Algoritmo 1: Busca Local Iterada

```
1. s = gera_solucao_inicial();
2. s* = busca_local(s);
3. Enquanto critério de parada não satisfeito
        3.1. s  = perturba(s*);
        3.2. s = busca_local(s);
        3.3. se custo de s melhor que custo de s* então
            3.3.1. s* = s;
        3.4. Fim Se;
 4. Fim Equanto
```

OBS: s^* representa a melhor solução conhecida até o momento

3 Um algoritmo BLI para o PCV

Seu trabalho neste laboratório é desenvolver uma Busca Local Iterada para o Problema do Caixeiro Viajante. As subseções que seguem fornecem sugestões de como você poderá implementar este algoritmo.

3.1 Representando os dados de entrada do problema

Os dados de entrada do problema consistem em uma matriz de distâncias simétrica (ANEXO I). Logo, pode-se utilizar um *array* bidimensional para representá-la.

3.2 Representando uma solução

Uma rota (ou solução para o PCV) pode ser representada por meio de um *array*. Por exemplo, considerando as cidades na ordem dada na Tabela 1 e seus respectivos códigos, um *array* que armazena a solução ótima do problema conteria os seguintes valores {1, 3, 10, 5, 9, 7, 2, 8, 6, 4}. Isto é, partindo-se da cidade 1 (Natal), deve-se ir para a cidade 3 (Parnamirim). Da cidade 3 (Parnamirim) para a cidade 10 (São José de Mipibu) e assim sucessivamente. Da última cidade, volta-se para a cidade inicial.

3.3 Calculando o custo de uma solução

Cada vez que nos deslocamos de uma cidade para a outra “pagamos” um custo, que é a distância. Desta forma, o custo da solução é o somatório das distâncias entre as cidades pelas quais se passa.

3.4 Obtendo uma solução inicial para o problema

Dado que uma solução para o problema pode ser representada por um *array*, como mostrado no item 3.2, pode-se gerar uma solução preenchendo-se este vetor com valores gerados aleatoriamente, sem que haja repetições. Estes valores devem corresponder ao código das cidades. A única restrição a este processo é que o primeiro elemento do *array* deve ser o ponto de partida.

3.5 Perturbando uma solução

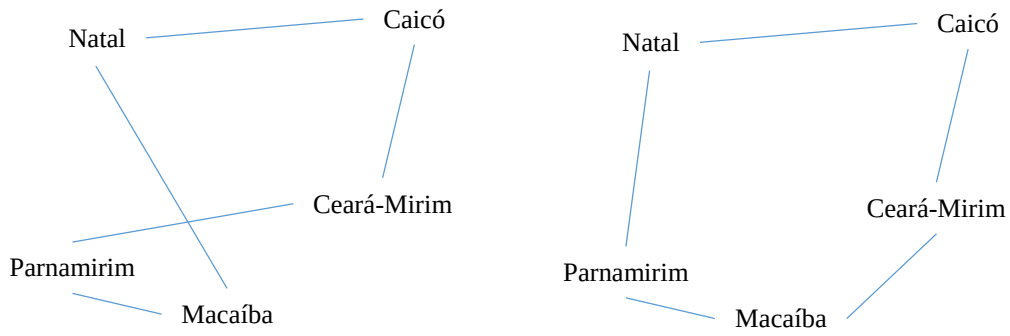
Perturbar, neste contexto, significa alterar aleatoriamente. Desta forma, pode-se perturbar uma solução dada para o PCV, que é representada por um *array*, trocando-se de posição alguns de seus valores (exceto o primeiro). Quanto mais valores são trocados, maior o grau de perturbação. Defina você mesmo um grau de perturbação. Por meio de testes, você pode ajustar este valor posteriormente.

3.6 Busca local

Uma busca local é um algoritmo cujo objetivo é fornecer um ótimo local. Isto é, obter a melhor solução dentre o conjunto de todas as soluções geradas a partir de uma solução inicial dada, utilizando-se para tal alguma estratégia movimentação dos elementos desta solução. Por exemplo, como estratégia de movimentação se pode considerar todas as possíveis formas de inverter a ordem de visitação de uma subrota.

Considerando-se a literatura do PCV, um dos algoritmos de busca local mais conhecidos é a $2-opt$ (Croes, 1958). Neste método, a ideia principal é alterar uma rota de forma que ela não passe por si mesma. Veja a Figura 1 para uma ilustração desta situação.

Figura 1: (a) Rota cruzada | (b) resultado 2-opt



Uma busca local 2-opt completa considera todas as possíveis combinações para troca de elementos na solução. Considerando-se que a solução do PCV está armazenada em um array `v` qualquer, o pseudocódigo em Algoritmo 1 mostra como a 2-opt altera uma rota.

Algoritmo 1: mecanismo básico de movimentação para a busca local 2-opt

```
Movimento2Opt(v, i, k) {
    nova_rota = array();
    1. insira em nova_rota os elementos de v[1] a v[i-1]
    2. insira em nova_rota os elementos de v[i] a v[k] em ordem inversa
    3. insira em nova_rota os elementos de v[k+1] até o final
    Devolva nova_rota;
}
```

O pseudo-código para a busca local 2-opt completa é dado no Algoritmo 2.

Algoritmo 2: busca 2-opt completa

```
Repita até que nenhuma melhora seja feita{
     início:
     melhor_custo = calcularCusto(rota_atual)
     para (i = 1; i < numero_cidades - 1; i++) {
         para (k = i + 1; k <numero_cidades; k++) {
             nova_rota = movimento2Opt(rota_atual, i, k)
             novo_custo = calcularCusto(nova_rota)
             if (novo_custo < melhor_custo) {
                 rota_atual = nova_rota
                 vá para início;
             }
         }
     }
 }
```

Agora é com você. Implemente o algoritmo de busca local iterada para o problema do caixeiro viajante. Sinta-se à vontade para melhorar o algoritmo descrito. O limite é a imaginação.

Referências

G. A. CROES (1958). A method for solving traveling salesman problems. Operations Res. 6 (1958) , pp., 791-812.

ANEXO I: DISTÂNCIA EM Km ENTRE AS 10 MAIORES CIDADES DO RIO GRANDE DO NORTE

	Natal	Mossoró	Parnamirim	São G. do Amarante	Macaíba	Ceará-Mirim	Caicó	Assu	Currais Novos	S. J. de Mipibu
Natal	0	281	20.9	18.1	29	39.7	273	214	187	38.4
Mossoró	281	0	269	263	254	255	192	73.7	207	286
Parnamirim	20.9	269	0	24.5	16.5	40.6	261	201	175	19.6
São Gonçalo do Amarante	18.1	263	24.5	0	9.3	21.5	255	196	169	42.2
Macaíba	29	254	16.5	9.3	0	30.8	246	187	160	33.4
Ceará-Mirim	39.7	255	40.6	21.5	30.8	0	276	188	190	63.4
Caicó	273	192	261	255	246	276	0	135	87.3	277
Assu	214	73.7	201	196	187	188	135	0	140	218
Currais Novos	187	207	175	169	160	190	87.3	140	0	192
São José de Mipibu	38.4	286	19.6	42.2	33.4	63.4	277	218	192	0

Fonte: http://distanciacidades.com/