Projeto e Análise de Algoritmos
2º Trabalho Prático

Tiago Alves Macambira

Belo Horizonte, 17 de junho de 2003

1 Exemplos práticos

Partes dessas respostas foram retiradas de [4].

1.1 Bin Packing

- Distribuicao de tarefas a professadores de acordo com os seus tamanhos e prioridades. Ness cenario, os processadores representam os *bins* e as tarefas com as suas respectivas prioridades os *pacotes* a serem colocados nos *bins*.
- Pretender distribuir incineradoras por diferentes localidades de um país, de forma a que não haja possibilidade de interferência dos resíduos de cada uma delas, o problema será de empacotamento. O objectivo será, por exemplo, a maximização dos resíduos que se poderá tratar.

1.2 Cobertura de Vértices

- e se pretender construir quartéis de bombeiros em pontos estratégicos de uma cidade, de forma a que todos os quarteirões vitais da cidade fiquem protegidos (ou seja, fiquem a uma distância inferior a um determinado limite), temos um problema de cobertura. O objetivo será a minimização do custo total de construção.

1.3 Mochila

- Suponha um ladrao que deseja roubar uma casa. Eh do maior interesse dele conseguir extrair do seu roubo, o maior lucro possivel com o menor esforso possivel. Para ele, o problema consiste em decidir quais bens levar de forma a ter menos trabalho para transporta-los mas ter o maior valor possivel. Nesse caso, os bens, seus valores e pesos sao os itens da mochila.

1.4 Subset Sum

- Considere que voce e correntista num banco e emitiu n os cheques $c_0, \ldots, c_n$. NO final so mes o banco informa apenas o total t descontado da sua conta devido a esses cheques. Quais cheques foram descontados?

1.5 Problema do Caixeiro Viajante

- Deseja-se fazer uma instalacao de uma rede local ponto-a-ponto (*token-ring*, por exemplo), onde cada computador liga-se a rede atravez de dois vizinhos. Deseja-se fazer com que essa instalacao tenha custo minimo e atenda a todos os pontos de rede da empresa.

2 Sobre o problema do Caixeiro viajante....

2.1 Prova que ele é NP-Completo

Inicialmente, mostremos que o problema do caixeiro viajante pertence a .NP. Dada uma intância do problema, podemos usar como certificado a seqüência de n vértices do percurso. O algoritmo de verificação checa se essa seqüência contem todos os vértices do grafo exatamente uma vez, soma o peso das arestas ligando esses vértices na ordem fornecida e verifica se a soma é no máximo k. Esse processo pode, sem sombra de dúvidas ser feito em tempo polinomial. Assim, provamos que ele pertence a NP.

Para provar que TSP é NP-Difícil, vamos gerar uma transformação polinomial do Ciclo de Hamilton para TSP. Seja $G = (V, E)$uma instância do problema do cliclo de Hamilton. Construimos a partir dele uma instância do problema do caxeiro viajante da seguinte maneira. Formamos um grafo completo $G' = (V, E')$onde $E = \{(i, j) : i, j \in V\}$,e definimos a função de custo para as arestas de G' da seguinte forma:

$$c(u, j) = \begin{cases} 0 & if \quad (i, j) \in E \\ 1 & if \quad (i, j) \notin E \end{cases}$$

A instância para o problema do caixeiro viajante será então $(G, c, 0)$, e esta foi facilmente gerada em tempo polinomial.

Agora mostramos que o prafo Gpossui um ciclo hamiltoniano se e somente se o grafo G'possuir um caminho de custo no máximo 0. Suponha que o grafo G tenha um cilco hamiltoniano h. Cada aresta em $h$pertence a E e por isso pssui custo 0 em G'. Assim, o caminho hé um caminho in G' com custo 0. Ja que o custo da arestas em E'são 0e , o custo do pergurso é exatamente 0. Assim, h'contêm apenas arestas em E. Concluimos então que hé um cliclo hamiltoniano em G. [1]

2.2 Algoritmo da força bruta

A minha implementacao apresentou os seguintes tempos:

numero de vertices	tempo de execucao em media
5	0.090s
6	0.115s
7	0.290s
8	0.670s
9	1.490s
10	8.430s
11	23.920s
12	1m4.870s
13	10m.880s

Depois de 13 vertices a execucao do algoritmo ficou por demais demorada.

Como um dos fatores que influenciaram os valores obtidos temos o uso de uma linguagem de script para a implementacao desse algoritmo[5] (uma implementacao em $C++$ foi gerada tambem mas apena utilizada para a submissao do trabalho). Essa escolha por si ja implica numa penalisacao do tempo de execucao do programa.

De qualquer forma, visto que esse problema e NP-Completo e que desejamos o valor para uma solucao otima desse problema, deparamos inevitavelmente com uma busca exaustiva atraves de forca-bruta para a solucao desse problema. Visto que a complexidade de uma busca em profuncidade nesse algoritmo 'e $O(n!)$, vemos que, independente de qualquer implementacao, o problema assume proporcoes gigantescas com pequenas variacoes do numero de vertices de entrada.

Apesar da implementacao usada para obter os valores dessa questao usar poda, afim de limitar o universo de procura e retornar um resultado mais rapidamente, ha de se obsevar que poda alguma poderia baixar significativamente os tempos de execucao desse problema: o universo de busca para ele 'e muuuito grande.

2.3 Heuristica

Foram implementamos duas heuristicas:

- Vizinho Mais proximo
- Arvore Geradora Minima

A literatura[1][2] comenta vastamente a complexidade de ambas.

2.3.1 Vizinho Mais Proximo

Nessa heuristica, conseguimos o seguinte caminho de custo 27384: 4 26 42 11 56 22 23 57 43 17 0 29 12 39 24 8 31 19 52 49 3 21 7 54 53 1 40 34 9 51 50 46 48 2 47 38 28 35 16 25 5 18 27 13 36 33 45 55 44 32 14 20 10 15 37 41 30 6

Nessa heuristica gulosa, iniciamos o percurso deo grafo em um dado vertice u, analisamos a lista de adjacencias desse vertice e escolhemos nela o vertice v cuja aresta ligando ao vertice u possui o menor custo. Seguimos repetindo esse processo, sempre escolhendo a aresta de menor valor que liga a um vertice nao visitado. Ora, podemos determinar se um vertice foi anteriormente visitado em $O(1)$, bastando utilizar *arrays* com $|V|$ entradas ou *bitsets*. Visitamos cada vertice uma vez e, a cada visita, percorremos toda a sua lista de adjacencias. Assim, toda aresta 'e visitada 2 vezes. Como para o problema do caixeiro viajante o grafo e completo, temos uma complexidade total $O(|E|)$.

A literatura[2] aponta para o fato de que a razao entre o valor encontrado por essa heuristica e a solucao otima 'e de

$$\frac{vizita}{otima} = \frac{1}{2} \limsup (\log_2 n) + \frac{1}{2}$$

2.3.2 Arvore Geradora Minima

Nessa heuristica, conseguimos o seguinte caminho de custo 28121: 45 33 14 36 13 27 5 18 25 16 35 28 2 34 9 50 46 48 42 26 4 22 11 56 23 43 17 0 29 12 39 24 8 31 19 52 49 3 57 51 40 1 53 54 21 7 47 38 10 15 37 30 6 41 20 32 44 55

Nessa implementacao:

- Modificamos o grafo de forma que ele respeite a desigualdade triangular
  - Completamos o grafo caso ele nao seja completo ($O(N^2)$)
  - verificamos todas as conbinacoes de 3 arestas, a procura da maior diferenca dentre aquelas entre os vertices que violam a desigualdade. ($O(n^3)$
  - somamos tal diferenca em todos os verices ($O(n)$)
- Executamos Prim
- Fazemos uma busca em profundidade nessa arvore ($O(n)$)

Note que o passo mais caro nesse processo e a verificacao da desigualdade triangular - a complexidade de Prim, se bem implementado, nao chega a tanto.

Ja o valor para o custo do caminho encontrado pode e relatado vastamente na ligeratura como sendo no maximo 2 vezes pior do que a solucao otima.[?][?].

2.4 Analise da Heuristica

Apos realisar teste com a heuristica do vizinho mais proximo, conseguimos plotar os seguintes valores:

heuristica	otimo	teto para a heuristica
48	37	74
21	16	32
63	40	80
58	34	68
54	39	78
26	21	42

Vemos que a heuristica se comportou melhor do que o teorico esperado dela, com resultados sempre melhores do que o dobro do otimo.

Referências

[1] "Introduction to Altorithms", Cormen et al, 1st Edition

[2] "Projeto de Algoritmos", Ziviani,NIvio. 1ª Edição. Editora. Pioneira

[3] ”Estruturas de Dados e seus Algoritmos”, Szwarcfiter, Jayme Luiz e Markenzon, Lilian. 2ª Edição. LTC Editora.

[4] ”Complementos de Inteligencia Artificial.”, Pedroso, Joao Pedro. Universidade do Porto, Portugal. http://www.dcc.fc.up.pt/ jpp/cia/

[5] ”The Python Programming Language”, http://www.python.org