Concepção e Análise de Algoritmos

Viagem Pela Europa

Parte 1

Ângela Cardoso, Felipe Galvão e Bruno Madeira

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

27 de Março de 2015

Conteúdo

Capítulo 1

Introdução

No âmbito da disciplina de Concepção e Análise de Algoritmos do Mestrado Integrado em Engenharia Informática e Computação, foi-nos proposta a resolução de um problema cuja formulação é frequentemente feita através da utilização de grafos.

O tema do nosso projeto é o planeamento de uma viagem pela Europa. Mais precisamente, o cliente pretende visitar várias cidades na Europa, deslocando-se de carro entre as cidades. Tem um tempo máximo predefinido para realizar a viagem, assim como uma classificação das cidades consoante o seu interesse, numa pontuação de 0 (pouco interessado) a 10 (muito interessado). Além disso, o cliente disponibiliza informação sobre o tempo de deslocação entre as cidades, assim como o tempo que demora a visitar cada uma delas. A solução a devolver ao cliente será uma lista ordenada de cidades a visitar, tendo em conta que começa e acaba na cidade onde reside.

Numa versão inicial do problema, o tempo para a viagem é limitado e, como tal, poderá não ser possível ir a todas as cidades, o objetivo é visitar as cidades mais interessantes. Numa outra versão, será calculado o tempo mínimo necessário para visitar todas as cidades, sendo devolvido o percurso mais eficiente.

Nesta primeira parte do projeto é apresentada uma formulação matemática do problema, que inclui as restrições assumidas. Serão ainda esboçadas possíveis resoluções, exatas e aproximadas, para as duas versões do problema, assim como propostas de análise do custo computacional dos algoritmos propostos.

Capítulo 2

Restrições Consideradas

Cada instância do nosso problema será representada por um grafo, em que os vértices representam as cidades a visitar e as arestas representam os percursos entre as cidades. Apesar de esta representação já contemplar uma série de aspetos do problema, existem ainda várias alternativas consoante os dados fornecidos e as restrições que sejam consideradas. A menos que algo seja dito explicitamente em contrário, à partida, o ideal será não impor restrições que limitem a aplicabilidade dos algoritmos desenvolvidos. Por outro lado, existem casos em que a limitação é recompensada pela eficiência dos algoritmos possíveis.

No caso do planeamento de uma viagem, há quatro questões que devem ser respondidas, por forma a identificar as eventuais restrições a considerar:

1. Dadas duas cidades, existe sempre um caminho que as une sem passar por nenhuma outra cidade da lista? Isto é, podemos considerar um grafo completo?

2. Dado um caminho que une as cidades A e B, existe alguma diferença entre ir de A para B ou de B para A? Isto é, teremos que considerar um grafo dirigido ou não?

3. No percurso devolvido é possível visitar a mesma cidade mais do que uma vez (além da cidade origem, claro)?

4. Podemos assumir que dadas três cidades A, B e C, o tempo de viagem de [A, C] nunca é maior que a soma dos tempos de [A, B] e [B, C]? Isto é, a desigualdade triangular é válida nos custos associados às arestas do grafo?

Apesar de não ser dito de forma explícita, iremos assumir que o grafo é completo, porque não se trata de uma verdadeira restrição para este problema específico. De facto, se não existisse um caminho direto entre as cidade A e C e fosse sempre necessário passar por C, poderíamos considerar uma nova aresta entre A e C, cujo tempo de percurso é a soma dos tempos [A, B] e [B, C]. Na prática isto corresponderia a ir da cidade A para a cidade C passando pela cidade B, mas sem a visitar.

Tipicamente, dentro de localidades, existem diferenças entre ir do ponto A para o ponto B ou fazer o percurso inverso. Já neste caso, à partida não seria muito restritivo assumir que não é necessário considerar arestas dirigidas, porque os tempos de deslocação serão relativamente grandes e pequenas diferenças devidas à orientação poderiam ser desprezadas. No entanto, uma vez que também é necessário considerar o tempo de visita de cada cidade a que chegamos, iremos considerar arestas dirigidos, cujo custo incluirá, além do tempo de viagem, o tempo de visita da cidade de destino. Ora, uma vez que os grafos considerados serão dirigidos, podemos perfeitamente permitir que os tempos sejam diferentes consoante se vá num sentido ou noutro. Convém no entanto notar que, ao considerar grafos completos dirigidos, teremos o dobro das arestas, em relação à versão do problema não dirigido. Este crescimento do grafo poderá ter um impacto negativo na eficiência dos algoritmos desenvolvidos.

Dado que o objetivo da primeira versão do problema é visitar as cidades mais interessantes dentro do tempo limite, visitar duas vezes a mesma cidade não é nada aconselhável, será melhor reservar esse tempo para visitar outra cidade diferente. Isto poderá não ser tão claro se considerarmos que, estando na cidade A, regressar a uma cidade já visitada B e depois visitar a cidade C é mais rápido do que ir diretamente de A para C. Ora, isto está obviamente relacionado com a resposta à quarta questão. Além disso, está também relacionado com a resposta à primeira questão. Se ir de A para C passando por B é mais rápido, porque não considerar que o caminho de A para C é precisamente A junção dos caminhos de A para B e B para C? Posto desta forma, não parece muito restritivo considerar que a resposta à quarta questão é sim, donde resulta que não há vale a pena visitar duas vezes a mesma cidade. Por outro lado, como veremos mais adiante, em termos de complexidade dos algoritmos, não há vantagem em impor a desigualdade triangular. Já a passagem pelo mesmo vértice mais do que uma vez, conduz a um tipo de problema bastante diferente. Sendo assim, vamos assumir que cada cidade é visitada no máximo uma vez e que a desigualdade triangular poderá não se verificar.

Em suma, temos uma grafo completo dirigido, do qual queremos extrair um percurso fechado que passa por cada vértice no máximo uma vez. Se passa por todos os vértices ou não, depende da versão do problema que estivermos a considerar.

Capítulo 3

Formulação do Problema

3.1 Dados de Entrada

- N - número de cidades que o cliente gostaria de visitar, incluindo a cidade de partida;
- T_{max} - tempo de que o cliente dispõe para a viagem na primeira versão do problema;
- $C[i]$, $(1 \leq i \leq N)$ - vetor com o nome de cada cidade a visitar, por exemplo $C[2] = Madrid$;
- $V[i]$, $(1 \leq i \leq N)$ - vetor com o valor (entre 0 e 10) atribuído a cada cidade, por exemplo $V[2] = 7$ representa a pontuação atribuída a $Madrid$;
- $T[i, j]$, $(1 \leq i, j \leq N)$ - matriz com os tempos de viagem entre cada par de cidades, quando $i = j$, $T[i, i]$ é o tempo de visita da cidade i.

Note-se que podemos referir-nos a cada cidade pelo seu índice, que é o mesmo no vetor C, no vetor V e na matriz T. Além disso, iremos assumir que a cidade 1 é a cidade de origem do cliente, que terá sempre que fazer parte do percurso, sendo $V[1] = 0 = T[1, 1]$.

3.2 Dados de Trabalho

Estes dados dependerão da versão do problema e do algoritmo usado para o resolver, pelo que serão apresentados numa próxima parte do projeto.

3.3 Dados de Saída

Para a primeira versão do problema, em que existe um tempo máximo para a viagem T_M e são consideradas as pontuações atribuídas às cidades, a solução final terá o formato seguinte:

- M - número de cidades do percurso obtido;
- $I[i]$, $(1 \leq i \leq M)$ - vetor com o índice de cada cidade do percurso, pela ordem em que é visitada, isto é, o percurso final será $I[1] -> I[2] -> \cdots -> I[M] -> I[1]$;
- V_{max} - valor do percurso obtido, ou seja, é a soma das pontuações das cidades do vetor P e, numa solução exata, é a pontuação máxima possível dentro do limite de tempo T_{max}.

Para a segunda versão do problema, em que serão visitadas todas as cidades e queremos qual a ordem de visita e o tempo necessário, a solução final terá o formato seguinte:

- T_F - tempo do percurso obtido, no caso de uma solução exata, é o menor tempo necessário para visitar todas as cidades;
- $I[i]$, $(1 \leq i \leq N)$ - vetor com o índice de cada cidade do percurso, pela ordem em que é visitada, isto é, o percurso final será $I[1] \to I[2] \to \cdots \to I[N] \to I[1]$.

3.4 Funções Objetivo

Mais uma vez, consoante a versão do problema, temos diferentes funções objetivo. Na primeira versão, queremos maximizar a soma de valores das cidades visitadas, garantindo que o tempo total do percurso não ultrapassa o tempo de viagem disponível. Isto é, assumindo que $I[M+1] = I[1] = 1$ é a cidade de origem, queremos obter um vetor I que cumpra as seguintes condições:

$$\mathcal{P}(I) : \sum_{1 \leq i \leq M} (T[I[i], I[i]] + T[I[i], I[i+1]]) \leq T_{max}; \tag{3.4.1}$$

$$\begin{gathered} \sum_{1 \leq i \leq M} V[I[i]] = V_{max} = \\ \max \left\{ \sum_{1 \leq j \leq \bar{M}} V[J[j]] : |J| = \bar{M}, J[j] \in \{1, 2, \ldots, N\}, \mathcal{P}(J) \text{ é válida} \right\}. \end{gathered} \tag{3.4.2}$$

Na segunda versão, queremos minimizar o tempo de visita a todas as cidades. Isto é, assumindo que $I[N+1] = I[1] = 1$ é a cidade de origem, queremos obter um vetor I que

cumpra a seguinte condição:

$$\sum_{1\leq i\leq N}(T[I[i],I[i]]+T[I[i],I[i+1]])=T_F=$$
$$\min\left\{\sum_{1\leq j\leq N}(T[J[j],J[j]]+T[J[j],J[j+1]]):J[j]\in\{1,2,\ldots,N\}\right\}. \tag{3.4.3}$$

Capítulo 4

Perspetivas de Solução

4.1 Backtracking Bruto

Criar/percorrer uma árvore para testar todas as combinações que se encontram dentro do tempo limite descartando as que ultrapassem o tempo limite. É guardado o melhor percurso (com maior "preferência" somada) e o seu respectivo valor a ser actualizados cada vez que encontrado um melhor. Caso não se pretenda o percurso óptimo o algoritmo pode parar ao encontrar uma solução que use todas as cidades.

4.2 Estratégia Mista

Esta abordagem procura um trajecto que cumpra as condições de limite de tempo e que tenha potencial para satisfazer a condição de maximização dos valores (preferências). Não apresentando uma solução óptima o algoritmo faz uso de heurística para conseguir obter uma solução aceitável em menos tempos que algoritmos mais precisos. O algoritmo funcionaria em 3 etapas distintas.

Primeiramente soma-se ao peso (tempo de visita) de cada vértice a o peso da aresta com maior peso para esse mesmo vértice (tempo de viagem). Assim sendo consideramos um conjunto de soluções $F[i]$, $(1 \leq i \leq L)$ para o problema em que:

$$peso(F) \equiv \sum_{1 \leq i \leq L} (\, T[F[i], F[i]] + \max\{T[F[i], j], (1 \leq j \leq N \wedge F[i] \neq j)\} \,)$$

Note-se para qualquer conjunto de solução possível I existe um conjunto C que contém os mesmos vértices (cidades) tal que:

$$\sum_{1 \leq j \leq M} (T[I[i], I[i]] + T[I[i], I[i+1]]) \leq peso(F) \tag{4.2.1}$$

assim sendo...

$$peso(F) \leq T_{max} \Rightarrow \exists I \mid \sum_{1 \leq j \leq M} (T[I[i], I[i]] + T[I[i], I[i+1]]) \leq T_{max} \tag{4.2.2}$$

Seguidamente é usado um algoritmo de branch and bound. Para este algoritmo teríamos:

1. O limite de tempo como o peso máximo;
2. As cidades (vértices) como itens;
3. A preferência dada pelo cliente como o ganho;
4. O peso seria dado pela soma do tempo de visita da cidade com o tempo encontrado na etapa 1. Assim sendo garante-se que o resultado obtido é uma combinação válida (que pode ser visitada dentro do tempo limite).

Uma vez já encontrada uma combinação de cidades que sabemos ser possível de visitar em qualquer ordem dentro do tempo dado podemos abordar o problema como um TSP para arranjar o melhor trajecto possível (que demora menos tempo) entre as cidades seleccionadas.

Capítulo 5

Métricas de Avaliação

Capítulo 6

Principais Dificuldades

Capítulo 7

Conclusão