C A P Í T U L O 3

O Problema do Caminho Mais Curto com um só Objectivo

1. Definição e formulação do problema

Os problemas de caminho mais curto com um só objectivo são fundamentais e frequentes quando se estudam problemas em redes, por exemplo de transportes ou de comunicações. Este problema surge quando se pretende determinar o caminho mais curto, mais barato ou mais fiável, entre um ou vários pares de nós de uma rede.

Estes problemas surgiram a partir de adaptações a uma grande variedade de problemas práticos, não só como modelos únicos mas também como subproblemas de problemas mais complexos. Por exemplo, surgiram nas indústrias de telecomunicações e de transportes sempre que se pretendia enviar uma mensagem, ou um veículo, entre dois locais geograficamente distantes, o mais rápido ou o mais barato possível.

O planeamento do tráfego urbano fornece um outro exemplo importante : os modelos utilizados para o cálculo do fluxo de tráfego padrão são problemas de optimização não linear, ou modelos de equilíbrio complexos. Contudo, a maioria das abordagens algorítmicas para determinar tráfegos urbanos padrão consiste, sob hipóteses simplificadoras, em resolver um grande número de problemas de caminho mais curto como subproblemas (um para cada par de nós origem–destino na rede) [1].

Existem três tipos de problemas de caminho mais curto :

- de um nó para outro,
- de um nó para todos os outros (árvore dos caminhos mais curtos),
- entre todos os pares de nós.

Os dois primeiros tipos são identificados como *problema de caminho mais curto de uma única origem* (ou apenas *caminho mais curto*) e o outro, como *problema de caminho mais curto entre todos os pares de nós*.

Sejam **s** e **t** dois nós de uma rede $G = (N, A, C)$, em que a cada arco está associado apenas um valor não negativo (*comprimento* do arco) — c_{ij} é o comprimrnto do arco (i, j). Seja c(p) o *comprimento* de um caminho **p** de **s** para **t** na rede G.

Com a resolução de um problema de caminho mais curto entre os nós **s** e **t**, na rede G, pretende-se determinar o caminho de valor mínimo existente em P; isto é, determinar $p \in P$, tal que $c(p) \leq c(q)$, para todo o $q \in P$ (P é o conjunto de todos os caminhos de **s** para **t**). Desta forma, pode-se formular este problema da seguinte forma :

$$\text{Minimizar} \quad Z = \sum_{i \in N} \sum_{j \in N} c_{ij}\, x_{ij}$$

sujeito a

$$\sum_{j \in N} x_{sj} = 1$$

$$\sum_{i \in N} x_{ij} - \sum_{k \in N} x_{jk} = 0, \qquad \forall j \in N - \{ s, t \}$$

$$\sum_{i \in N} x_{it} = 1$$

em que,

$$x_{ij} = \begin{cases} 1, & \text{se (i, j) pertence ao caminho} \\ 0, & \text{se (i, j) não pertence ao caminho} \end{cases}$$

Algumas observações relacionadas com este tipo de problemas :

- o comprimento de um caminho é maior do que o de qualquer dos seus subcaminhos;
- qualquer subcaminho de um caminho mais curto, é ele próprio um caminho mais curto (princípio da optimalidade);
- para uma rede com **n** nós, qualquer caminho mais curto tem no máximo **n-1** arcos (num caminho não existem nós repetidos).

Para se analisar convenientemente a eficiência dos algoritmos, deve ser estudada a ordem de complexidade de cada um deles e compará-las. A ordem de complexidade de um algoritmo depende da quantidade de operações que ele executa, quando se considera o pior dos casos. Desta forma, a ordem de complexidade pode traduzir o tempo de execução do

algoritmo, para aquele caso. Portanto, a ordem de complexidade será o limite superior de tempo que é necessário para executar qualquer problema de caminho mais curto.

2. Algoritmos para resolver problemas de caminho mais curto

Existem vários algoritmos eficientes para resolver problemas de caminho mais curto com um só objectivo, sendo os mais conhecidos os algoritmos de Dijkstra, de Ford e de Floyd. Os dois primeiros aplicam-se a problemas de caminho mais curto de um nó inicial, **s**, para um nó final, **t** (ou para todos os outros), e baseiam-se num processo de rotulação dos nós; o último aplica-se a problemas de caminho mais curto entre todos os pares de nós.

2.1. Algoritmo de Dijkstra

Este algoritmo, que foi apresentado por Dijkstra, só pode ser aplicado a redes a cujos arcos estão associados apenas valores não negativos. Neste algoritmo, é suposto existir pelo menos um caminho entre **s** e qualquer outro nó.

O algoritmo baseia-se num processo de rotulação dos nós da rede e classificação dos respectivos rótulos. A cada nó **i** é atribuído um rótulo $[\xi_i, \pi_i]$ que pode ser permanente (fixo) ou temporário. Isto é,

$[\xi_i, \pi_i]$ permanente (o caminho mais curto de **s** para **i**)

$\xi_i \leftarrow$ nó que antecede **i** no caminho mais curto de **s** para **i**

$\pi_i \leftarrow$ comprimento do caminho mais curto de **s** para **i**

$[\xi_i, \pi_i]$ temporário (um caminho mais curto de **s** para **i**)

$\xi_i \leftarrow$ nó que antecede **i** no melhor caminho, até ao momento, de **s** para **i**

$\pi_i \leftarrow$ comprimento do melhor caminho, até ao momento, de **s** para **i**

$[\xi_i, \pi_i] = [-, \infty]$ indica que ainda não foi encontrada qualquer caminho de **s** para **i**.

O rótulo temporário de um nó representa um limite superior da distância mais curta de **s** para esse nó, uma vez que o caminho que lhe está associado pode ser ou não o mais curto.

O algoritmo consiste num processo de fixação dos rótulos dos nós da rede, começando pelo **s**, de uma forma ordenada segundo as distâncias de cada nó a **s**. Em cada iteração é escolhido o nó **i** com rótulo temporário com menor valor de π, que se torna permanente, para depois se varrerem todos os nós adjacentes a **i** (que tenham rótulos temporários), para que os seus rótulos sejam actualizados. O algoritmo termina quando não existirem nós com rótulos

temporários (caminho mais curto do nó **s** para todos os outros) ou quando o rótulo do nó **t** passar a permanente (caminho mais curto do nó **s** para o nó **t**).

Quando o algoritmo terminar, o comprimento do caminho mais curto entre **s** e **j** é π_j e o caminho determina-se percorrendo (em sentido inverso) a primeira parte dos rótulos (ξ) de **j** até **s**, da seguinte forma :

Caminho $\leftarrow$ { j }
i $\leftarrow$ j
Enquanto i $\neq$ s **Fazer**
    i $\leftarrow \xi_i$
    Caminho $\leftarrow$ Caminho $\cup$ { i }

A versão do algoritmo de Dijkstra que se descreve em Algoritmo 1 é a original e foi adaptada a partir de Ahuja et al. [1].

$[\xi_s, \pi_s] \leftarrow [s, 0]$ *(caminho mais curto de **s** para **s** tem comprimento 0 e **s** é o nó que antecede **s**)*
$[\xi_i, \pi_i] \leftarrow [-, \infty]$, $\forall$ i $\in$ N – { s } *(o caminho mais curto de **s** para **i** é desconhecido)*
R $\leftarrow$ N *(R = conjunto de nós com rótulo temporário)*
$\overline{R} \leftarrow \varnothing$ *($\overline{R}$ = conjunto de nós com rótulo permanente)*
Enquanto R $\neq \varnothing$ (ou $\overline{R} \neq$ N) **Fazer**
    k $\leftarrow$ nó de R tal que π_k é mínimo *($k : \pi_k = min \{ \pi_x \}, x \in R$)*
    R $\leftarrow$ R – { k } *(k deixou de ter rótulo temporário)*
    $\overline{R} \leftarrow \overline{R} \cup$ { k } *(k passou a ter rótulo permanente)*
    Para todo o j $\in$ N tal que (k, j) $\in$ A **Fazer**
        Se $\pi_k + c_{kj} < \pi_j$ **Então**
            $\pi_j \leftarrow \pi_k + c_{kj}$
            $\xi_j \leftarrow$ k

Algoritmo 1. Caminho mais curto : Dijkstra.

Para se contabilizar o número de operações que serão efectuadas neste algoritmo, pode-se fazer uma análise segundo duas componentes : selecção do nó com rótulo temporário mínimo e actualização dos rótulos. Para tal, considere-se uma rede orientada completa com **n** nós e **m** arcos (pior dos casos) — **m = n.(n–1)**.

Relativamente à primeira componente, são seleccionados **n** nós e a partir de cada um deles são varridos os (n–1) nós (associados aos (n–1) arcos que saem de cada nó), excepto aqueles que já têm rótulo permanente; ou seja, a partir do k-ésimo nó seleccionado são varridos n–k nós ((n–1) – (k–1)). Portanto, o número total de operações é : $n + (n–1) + (n–2) + \cdots + 1 = (n^2 + n) / 2 = O(n^2)$ (ou seja, da ordem de complexidade de n^2 operações).

Em relação à segunda componente, como cada arco só é analisado uma única vez, então esta operação é executada **m** vezes; logo, a ordem de complexidade é O(m).

Desta forma, a ordem de complexidade do algoritmo é de $O(n^2) + O(m) = \mathbf{O(n^2)}$ porque $m < n^2$. Este limite é o melhor que se pode obter em problemas que exijam redes completas ou muito densas para a representação dos seus dados no computador.

Relativamente ao espaço de memória necessário à execução deste algoritmo, é preciso armazenar uma matriz quadrada de ordem **n** e três vectores de dimensão **n**. Ou seja, são necessários, pelo menos, $\mathbf{n^2 + 3.n}$ espaços de memória para guardar informação — $\mathbf{O(n^2)}$.

2.2. Outras versões (mais eficientes) do algoritmo de Dijkstra

O facto das redes envolvidas na grande parte dos problemas reais serem esparsas, leva a que se possa utilizar versões mais eficientes deste algoritmo — as que exploram esta característica.

Pelo que se verificou em relação à ordem de complexidade do algoritmo de Dijkstra, a operação mais dispendiosa é a de selecção de nós. Desta forma, todos as versões que visem o melhoramento deste algoritmo têm que se debruçar sobre este aspecto.

Surgiram então diversas versões baseadas na ordenação dos rótulos (conjunto R), em função dos valores de π : sempre que o rótulo de um nó é alterado, a ordenação do conjunto terá que ser actualizada, pois pode acontecer que a ordem daquele nó também seja alterada. Por outro lado, é desnecessário considerar o conjunto dos nós com rótulos permanentes ($\overline{R}$), uma vez que é o complementar de R (em $\mathbb{N}$).

Portanto, das várias versões que surgiram, destacam-se as seguintes :

a) utilização de uma lista, em que os seus elementos são ordenados crescentemente segundo o valor de π e o elemento escolhido (associado ao nó, cujo rótulo passa a permanente) é o que se encontra à cabeça;

b) utilização de duas listas ligadas (lista duplamente ligada), em que os processos de ordenação e escolha são os mesmos do anterior, mas com esta representação torna-se mais rápido, em particular a ordenação;

c) utilização de tabelas de "hashing" (HEAP — lista de listas), em que os elementos que contém a mesma característica são colocados num mesmo grupo (lista) e em que estes grupos são ordenados pelas diferentes características (lista); depois, o elemento escolhido é o que se encontra à cabeça da primeira lista. Por exemplo, dado um valor C, é construída uma lista de C elementos, em que cada elemento contem uma lista de nós, colocados por qualquer ordem, cujo valor do resto da divisão inteira de π por C é igual : 0 (1ª lista), 1 (2ª lista), 2 (3ª lista), …, C–1 (C-ésima lista).

2.3. Algoritmo de Dijkstra Inverso

No algoritmo de Dijkstra normal, determina-se o caminho mais curto do nó **s** para qualquer outro nó $\mathbf{j} \in N - \{ s \}$. No entanto, se se pretender determinar o caminho mais curto de qualquer nó $\mathbf{j} \in N - \{ t \}$ para um nó final **t**, utiliza-se uma pequena modificação do algoritmo de Dijkstra, que consiste em manter a distância ρ_j associada a cada nó **j**, que é um limite superior para o comprimento do caminho mais curto do nó **j** para o nó **t**. Todo o restante processo é igual ao utilizado para o algoritmo normal.

2.4. Algoritmo de Ford

Ao contrário do algoritmo de Dijkstra, este algoritmo, que foi proposto por Ford, pode ser aplicado a redes cujos arcos têm associado valores negativos, não podendo contudo existir circuitos de valor negativo. Também neste algoritmo é suposto existir pelo menos um caminho entre **s** e qualquer outro nó.

Este algoritmo consiste em rotular os nós da rede, corrigindo-os as vezes que for necessário, até que todos os rótulos satisfaçam a seguinte condição (de optimalidade) :

$$\pi_j \leq \pi_i + c_{ij}\ , \text{ para todo o } (i, j) \in A,$$

em que π_i corresponde ao valor do caminho mais curto de **s** para **i** e c_{ij} ao valor do arco (i, j).

Neste algoritmo, os rótulos dos nós têm o mesmo significado dos rótulos utilizados no algoritmo de Dijkstra, só que agora os rótulos só se tornam definitivos após terminar a execução do algoritmo. Ou seja, nos estados intermédios do algoritmo apenas existem rótulos temporários.

Tal como acontece com o algoritmo de Dijkstra, também neste, após a conclusão da execução do algoritmo, o caminho mais curto entre os nós **s** e **j** determina-se percorrendo os rótulos (em sentido inverso) desde **j** até **s**.

Em Algoritmo 2, encontra-se a descrição da versão original do algoritmo de Ford, que foi adaptada a partir de Ahuja et al. [1].

$[\xi_s, \pi_s] \leftarrow [s, 0]$ *(caminho mais curto de **s** para **s** tem comprimento 0 e **s** é o nó que antecede **s**)*

$[\xi_i, \pi_i] \leftarrow [-, \infty]$, $\forall\ i \in N - \{ s \}$ *(o caminho mais curto de **s** para **i** é desconhecido)*

$R \leftarrow \{ s \}$

Enquanto $R \neq \varnothing$ **Fazer**

  $R \leftarrow R - \{ i \}$, para i qualquer

  Para todo o arco (i, j) **Fazer**

    Se $\pi_j > \pi_i + c_{ij}$ **Então**

      $\pi_j \leftarrow \pi_i + c_{ij}$

      $\xi_j \leftarrow i$

      Se $j \notin R$

      Então $R \leftarrow R \cup \{ j \}$

Algoritmo 2. Caminho mais curto : Ford.

Considere-se uma rede orientada completa com **n** nós e **m = n.(n–1)** arcos. Para se determinar a ordem de complexidade deste algoritmo, apenas é necessário contabilizar o número de iterações do ciclo, já que em cada iteração são efectuadas 2 operações (de actualização dos rótulos). O ciclo é executado no máximo **n.m** vezes, pois todos os nós e todos os arcos são analisados. Logo, a ordem de complexidade deste algoritmo é **O(n.m)**.

Relativamente ao espaço de memória, o algoritmo necessita de uma matriz quadrada de ordem **n** e de dois vectores de dimensão **n**. Portanto, no total são necessários $\mathbf{n^2 + 2.n}$ espaços de memória para guardar informação — $\mathbf{O(n^2)}$.

2.5. Outras versões (mais eficientes) do algoritmo de Ford

Surgiram posteriormente diversas versões relacionadas com a forma de manipular a lista dos nós sucessores dos arcos que são analisados, das quais se destacam as seguintes :

a) LIFO ("Last In First Out") : a inserção e a remoção de elementos é efectuada na cabeça da lista (o último nó inserido é o primeiro a ser removido) — Pilha (pesquisa em profundidade);

b) FIFO ("First In First Out") : a inserção de um elemento é efectuada pela cauda e a remoção pelo topo (o primeiro nó inserido é o primeiro a ser removido) — Fila (pesquisa por níveis);

c) DEQUEUE : as remoções fazem-se na cabeça da lista e as inserções na cabeça ou na cauda, conforme uma condição é ou não satisfeita (por exemplo : **se** π_j foi alterado **então**, se **j** já pertenceu à lista então j é inserido no topo senão j é inserido na cauda);

d) SCALING : a lista é manipulada como um "Dequeue", utilizando o seguinte critério para decidir se um elemento é inserido no topo ou na cauda : **se** a diminuição de π_j for maior que um dado δ **então**, se **j** já pertenceu à lista então **j** é inserido no topo senão **j** é inserido na cauda.

2.6. Algoritmo de Floyd

Ao pretender-se determinar o caminho mais curto entre todos os pares de nós, pode-se aplicar o algoritmo de Dijkstra (ou Ford) **n** vezes, utilizando cada nó, sucessivamente, como origem. No entanto, existem outros algoritmos para resolver este problema, como é o caso do algoritmo de Floyd, desde que não haja circuitos negativos; neste algoritmo, os arcos podem ter associados valores positivos ou negativos.

Um arco (i, j) designa-se por **arco básico** se constituir o caminho mais curto entre os nós **i** e **j**. Um caminho mais curto entre quaisquer dois nós da rede será totalmente constituído por arcos básicos (embora existam arcos básicos não pertencentes ao caminho mais curto).

O algoritmo de Floyd utiliza a matriz D, de ordem **n**, das distâncias directas mais curtas entre os nós. Os nós que não são adjacentes (não existe arco a ligá-los) têm associado uma distância directa infinita. Como os nós são (por convenção) adjacentes a eles próprios, têm associado um arco de comprimento 0 (zero). Os ciclos próprios são ignorados.

Entre cada par de nós não ligados por um arco básico é criado um arco, através de um processo identificado por "tripla ligação" :

$$d_{ij} \leftarrow \min \{ d_{ij} , \ d_{ik} + d_{kj} \}$$

Fixando um k, a "tripla ligação" é efectuada para todos os nós i, j ≠ k.

Após efectuar a "tripla ligação" para cada k ∈ N e i, j ∈ N – { k }, a rede (na matriz D alterada ao longo deste processo) é apenas constituída por arcos básicos. Logo, o valor associado a cada arco dirigido (i, k) é o caminho mais curto entre aqueles nós.

Para conhecer todos os nós intermédios num dado caminho, mantém-se paralelamente uma matriz P, de ordem **n**, onde o elemento P_{ij} representa o primeiro nó intermédio entre os nós **i** e **j**. No final deste algoritmo, a matriz D corresponde à matriz das distâncias mais curtas entre qualquer par de nós. Em Algoritmo 3 encontra-se a descrição deste algoritmo, que foi adaptado a partir de Ahuja et al. [1].

```
D ← matriz das distâncias directas
P_ij ← j , ∀ i, j ∈ N
Para k = 1, ..., n Fazer
    Para i, j = 1, ..., n tal que i, j ≠ k Fazer
        Se D_ij > D_ik + D_kj Então
            D_ij ← D_ik + D_kj
            P_ij ← P_ik
```

Algoritmo 3. Caminho mais curto : Floyd.

Para se determinar a ordem de complexidade deste algoritmo, considere-se uma rede completa, com **n** nós e **m** = n.(n–1) arcos (pior dos casos). Neste caso, a quantidade de operações que são efectuadas depende do número de iterações dos ciclos, uma vez que em cada iteração são efectuadas, no máximo, três operações (incluindo o teste). Portanto, para este caso, o ciclo externo é executado **n** vezes e o interno **2.(n–1)**, o que perfaz um total de $n.2.(n–1) = 2.n^2 – 2.n =$ **$O(n^2)$**.

Relativamente ao espaço de memória exigido, são necessárias duas matrizes quadradas de ordem **n**; logo, são precisos **$2.n^2$** espaços de memória para guardar informação — **$O(n^2)$**.

3. Árvore dos caminhos mais curtos

Quando se pretende determinar os caminhos mais curtos de um nó origem para todos os outros, está-se perante um problema de caminho mais curto. Para armazenar os dados referentes a estes caminhos, apenas é necessário utilizar (n–1) espaços de memória, em que **n** é quantidade nós da rede. Este resultado é consequência do facto de se poder determinar sempre uma árvore com raiz no nó origem, à qual pertencem todos os nós da rede para os quais exista caminho desde o nó origem, com a seguinte propriedade : o único caminho da origem para qualquer nó é o caminho mais curto para aquele nó.

Portanto, para se determinar a árvore dos caminhos mais curtos de um nó origem para todos os outros numa qualquer rede, basta aplicar um dos dois primeiros algoritmos de caminho mais curto, apresentados anteriormente.

4. O problema de determinar os k caminhos mais curtos

Este problema coloca-se quando é necessário conhecer, não só o caminho mais curto entre dois nós de uma rede, mas também o 2^o, o 3^o, ..., o k-ésimo caminho mais curto, tal que o valor do **k**-ésimo caminho é menor ou igual ao valor do **j**-ésimo caminho, com j > k. Pretende-se então determinar uma ordenação (crescente) dos caminhos mais curtos, os quais constituem caminhos mais curtos alternativos ao melhor caminho.

Os problemas de determinar os k caminhos mais curtos pode ser aplicado a casos em que se pretenda determinar o caminho mais curto, mas obedecendo a restrições complexas, em que a melhor solução pode não ser o caminho mais curto. Portanto, determinando-se sucessivamente os caminhos por ordem crescente dos seus comprimentos, pode-se chegar ao caminho mais curto que obedece às restrições impostas.

Em problemas de circulação de veículos interessa, muitas vezes, conhecer caminhos alternativos ao caminho mais curto, em virtude de existir congestionamento ou corte em qualquer troço daquele caminho.

Em problemas de encaminhamento de chamadas, pode interessar conhecer caminhos alternativos ao mais curto, por exemplo, quando alguma ligação deste caminho está saturada.

Um outro tipo de problemas que pode utilizar o algoritmo dos k caminhos mais curtos, está relacionado com o problema do caminho mais curto com múltiplos objectivos.

Formalmente, pode-se definir o problema de determinar os k caminhos mais curtos da seguinte forma :

Seja $G = (N, A, C)$ uma rede com **n** nós e **m** arcos. Considere-se $c : P \rightarrow \Re$ tal que $p \rightarrow c(p) = \sum_p c_{ij}$ é um valor real definido sobre P, que associa um valor a cada caminho de **s** para **t** em G. Dado um inteiro positivo k, pretende-se determinar um conjunto $P^{(k)} = \{ p_1, ..., p_k \} \subset P$ tal que

1) p_i é determinado antes de p_{i+1} , para qualquer $i \in \{ 1, ..., k-1 \}$

2) $c(p_i) \leq c(p_{i+1})$, para qualquer $i \in \{ 1, ..., k-1 \}$

3) $c(p_k) \leq c(q)$, para qualquer $q \in P - P^{(k)}$.

Existem alguns algoritmos para tratar este problema, dos quais serão referidos três deles, de acordo com a sua importância na evolução dos algoritmos :

- baseado no Princípio da Optimalidade — o clássico algoritmo de Dreyfus [12],
- baseado no aumento da rede — desenvolvido por Martins [17],
- baseado na determinação de nós desvios — desenvolvido por Martins et al. [20].

Enquanto que os dois primeiros determinam os k trajectos mais curtos, o último determina apenas caminhos (um caminho é um trajecto sem ciclos).

4.1. Algoritmo de Dreyfus

O objectivo deste algoritmo é determinar os k trajectos mais curtos entre um nó origem (assume-se que é o nó 1) e os restantes (n–1) nós da rede — Algoritmo 4.

Em Algoritmo 4 (em Rodrigues [21]), atenda-se à seguinte notação :

- $u_j[h]$ é o comprimento do h-ésimo trajecto mais curto do nó origem para o nó j;
- μ (i, j, h) é a quantidade de trajectos nos quais (i, j) é o arco final, no conjunto dos h trajectos mais curtos ($\{ 1^o, 2^o, ..., h\text{-ésimo} \}$) do nó origem para o nó j.

Neste algoritmo os nós **j** devem ser processados na ordem da quantidade de arcos existentes nos respectivos trajectos mais curtos a partir da origem, de modo a que o valor $u_i[h+1]$ seja conhecido na altura em que se pretenda calcular $u_j[h+1]$, se $(i, j) \in A$.

$u_1[1] \leftarrow 0$ (*por convenção, o nó origem é o 1*)

Para todos os arcos (i, j) **Fazer**

μ (i, j, 0) $\leftarrow 0$

Determinar a árvore dos caminhos mais curtos a partir do nó origem

Para os nós j = 2, 3, ..., n **e** i = 1, 2, ..., n, com i $\neq$ j **Fazer**

Se i é predecessor de j na árvore dos caminhos mais curtos **Então**

μ (i, j, 1) $\leftarrow \mu$ (i, j, 0) + 1

Senão

μ (i, j, 1) $\leftarrow \mu$ (i, j, 0)

Para h =1, ..., k **Fazer**

Para todo o nó j **Fazer**

Se (i, j) é o arco final do h-ésimo trajecto mais curto da origem até j **Então**

μ (i, j, h) $\leftarrow \mu$ (i, j, h–1) + 1

Senão

μ (i, j, h) $\leftarrow \mu$ (i, j, h–1)

(i, j) $\leftarrow$ último arco do trajecto mais curto da origem para j

$$u_j[h] \leftarrow \min_i \left\{ u_i\left[\mu(i, j, h-1)+1\right] + c_{ij} \right\}$$

Algoritmo 4. Determinar os k trajectos mais curtos : Dreyfus.

Em termos de complexidade, a quantidade de operações que este algoritmo efectua é da ordem de **O(k.n.log n)**, não contabilizando as operações associadas ao cálculo da árvore dos caminhos mais curtos, nem a atribuição de valores iniciais a u e μ.

Este algoritmo necessita de **k.n** espaços de memória para guardar a informação associada a u e m espaços para a associada a μ — em cada iteração é preciso guardar o valor de μ (i, j, h) para cada arco (i, j) — para além do espaço necessário para armazenar a rede. Portanto, o algoritmo necessita de um total de $\mathbf{n^2 + k.n + m}$ espaços de memória para armazenar informação — $\mathbf{O(n^2)}$.

4.2. Algoritmo baseado no aumento da rede

O desenvolvimento deste tipo de algoritmos deveu-se, principalmente, a Martins [17], cuja ideia inicial tem sofrido melhorias ao longo dos tempos. Este tipo de algoritmo consiste em aumentar a rede, sempre que se encontre um trajecto, em que esta alteração está directamente relacionado com o caminho que foi encontrado.

Considerando que se pretende determinar os k trajectos mais curtos entre os nós **s** (origem) e **t** (destino) na rede $G(N, A, C)$, o algoritmo começa por adicionar um "super nó origem" S, um "super nó destino" T e dois arcos de custo nulo, (S, s) e (t, T). Desta forma, em qualquer trajecto de S para T existe um subtrajecto de **s** para **t**, cujos comprimentos são iguais, pois (S, s) e (t, T) têm comprimentos nulos.

O primeiro passo do algoritmo consiste em determinar a árvore dos caminhos mais curtos do nó **s** para todos os outros, a partir da qual se determina o trajecto mais curto de **s** para **t**. Depois, e em cada iteração **h** do algoritmo, a rede é alterada adicionando mais nós (alternativos) e mais arcos (fictícios), sobre a qual é determinado o trajecto mais curto, que coincide com o (h+1)-ésimo trajecto mais curto da rede original.

Assim, para se determinar a sequência dos k trajectos mais curtos, $\{ p_1, p_2, ..., p_k \}$, o algoritmo constrói uma sequência de k nós $\{ t, t', t'', ..., t^{(k-1)'} \}$, em que cada um representa o nó terminal **t** e o nó alternativo ao trajecto mais curto. Além disso, o k-ésimo trajecto mais curto pode ser determinado desde $t^{(k-1)'}$ até **s**, andando para trás, substituindo todos os nós alternativos pelos respectivos nós originais.

Em Algoritmo 5 descreve-se a versão mais recente deste algoritmo (MS), que foi apresentada por Martins e Santos [19]. Para qualquer nó $x \in N$ considere a seguinte notação :

Pred(x) é o nó que antecede x em p — $(Pred(x), x) \in p$

Suc(x) é o nó sucessor de x em p — $(x, Suc(x)) \in p$

π_x é o valor da distância mais curta de **s** para **x**,

$I(x) = \{ (i, x) : (i, x) \in A \}$ é o conjunto dos arcos que chegam a **x**,

$T(x) = \{ i \in N : (i, x) \in I(x) \}$ é o conjunto dos nós cauda dos arcos de I(x),

x′ é o nó alternativo de x,

$\bar{x}$ é o nó da rede original que corresponde ao nó x,

$x^{(h)'}$ é o h-ésimo nó alternativo de x (x tem h nós alternativos) — $x^{(0)'} \equiv x$, $\forall\, x \in N$

Determinar a árvore dos caminhos mais curtos a partir de **s** em G

$p_1 \leftarrow$ caminho mais curto de **s** para **t** na árvore dos caminhos mais curtos

$h \leftarrow 1$

$p \leftarrow p_1$

Enquanto existe um trajecto alternativo a p **e** $h < k$ **Fazer**

$u \leftarrow$ o primeiro nó de p tal que a $\overline{u}$ chega mais do que um arco *{ 1ª fase }*

Se $u' \notin N$ **Então**

$(N, A) \leftarrow (N, A) \cup \{ u' \}$

$T(\overline{u}) = T(\overline{u}) - \{ Pred(u) \}$

$I(\overline{u}) = I(\overline{u}) - \{ (Pred(u), \overline{u}) \}$

$\pi_{u'} \leftarrow \min \{ \pi_x + dist(x, \overline{u}) : (x, \overline{u}) \in I(\overline{u}) \}$

$v \leftarrow Suc(u)$

Senão

$v \leftarrow$ primeiro nó que segue $u \in p$ tal que $u' \in N$

Para cada $w \in \{ v, ..., t^{(h-1)'} \}$ **Fazer** *{ 2ª fase }*

$(N, A) \leftarrow (N, A) \cup \{ w' \}$

$T(\overline{w}) = T(\overline{w}) - \{ Pred(w) \} \cup \{ Pred(w)' \}$

$I(\overline{w}) = I(\overline{w}) - \{ (Pred(w), \overline{w}) \} \cup \{ (Pred(w)', \overline{w}) \}$

$\pi_{w'} \leftarrow \min \{ \pi_x + dist(x, \overline{w}) : (x, \overline{w}) \in T(\overline{w}) \}$

$p \leftarrow$ trajecto mais curto de s para $t^{(h)'}$ em G

$h \leftarrow h + 1$

Algoritmo 5. Determinar os k trajectos mais curtos : baseado no aumento da rede (MS).

Em termos de complexidade, a quantidade de operações que esta versão do algoritmo efectua é da ordem de **O(k.m)**, não tendo em conta as operações associadas ao cálculo da árvore dos caminhos mais curtos.

Comparando com a versão original[1], a quantidade de operações efectuadas não foi melhorado. No entanto, relativamente ao espaço de memória necessário para o algoritmo ser

[1] A versão origial deste algoritmo encontra-se descrita em Martins e Santos [19].

executado, enquanto que a quantidade de operações na versão inicial é da ordem de **O(k.n + k.m)**, a última versão apenas é da ordem de **O(k.n + m)** — ou de **O(k.n)** se k > m/n.

Note-se que esta questão do espaço de memória é muito importante, já que permite a sua aplicação a problemas que envolvam redes muito grandes.

Comparando a versão aqui apresentada (a última) com a original, verifica-se que enquanto na versão original as quantidades de nós e de arcos sofrem um significativo aumento, uma vez que quando se adiciona um nó alternativo x', adicionam-se tantos arcos quantos os que chegam a x menos 1 (se $(y, x) \in A$ e $(y, x) \notin p$ então (y, x') é adicionado à rede), na versão aqui apresentada, apenas o número de nós é aumentado, já que o número de arcos pode ser mantido (ou até ligeiramente decrementado), pois sempre que se adiciona um nó alternativo x', retira-se o arco de **p** que chega a x e, ou não se adiciona qualquer arco (1ª fase) ou apenas se adiciona um arco que liga o nó alternativo de Pred(x) a x (2ª fase).

4.3. Algoritmo baseado na determinação de nós desvios

Este algoritmo, que foi apresentado por Martins et al. [20], utiliza a mudança de variável proposta por Eppstein nos seus algoritmos para problemas sem restrições e o conceito de nó desvio de um trajecto.

Em termos gerais, a iteração **h** deste algoritmo consiste em determinar os desvios que podem dar origem a caminhos, os quais são calculados a partir do h-ésimo caminho mais curto, que foi encontrado nesta versão.

A mudança de variável é apresentada no seguinte resultado : seja ρ_i a distância mais curta de **i** para **t** e $\bar{c}_{ij} = \rho_j - \rho_i + c_{ij}$, $\forall (i, j) \in A$. Então, $\bar{c}(p) = \sum_p \bar{c}_{ij} = \rho_t - \rho_s + c(p)$, $\forall p \in P$.

Para resolver o problema da árvore dos caminhos mais curtos de cada nó $\mathbf{i} \in N - \{ t \}$ para **t**, utiliza-se o algoritmo de Dijkstra Inverso (ver secção 2.3 deste capítulo).

Por outro lado, o conjunto de arcos que saem de cada nó **i** tem de ser ordenado por ordem decrescente dos valores de $\bar{c}_{ij}$, de tal modo que o último arco pertença ao caminho mais curto de **i** para **t**.

Considere-se a seguinte definição de nó desvio de um caminho. Seja **q** o caminho mais curto de **s** para **i** na árvore de todos os trajectos já determinados. Para todo o k > 1, o nó **i** é o nó desvio de p_k se $q \subset p_k$ e for o subcaminho com o maior número de nós tal que $q \subset p_h$, com $h \in \{ 1, ..., k-1 \}$. Por conveniência, o nó **s** é o nó desvio de p_1. Considere-se o seguinte

exemplo : p_1 = [1, 2, 3, 4, 5], p_2 = [1, 2, 4, 5], p_3 = [1, 2, 3, 5] e p_4 = [1, 2, 4, 3, 5] são caminhos de uma rede qualquer. O nó desvio de p_4 é o nó 3, pois q = [1, 2, 3] é o subcaminho com origem em 1 com o maior número de nós entre todos os três caminhos dados.

Em Algoritmo 6 descreve-se a versão mais recente deste algoritmo (MPS), que foi apresentada por Martins et al. [20].

Determinar a árvore dos caminho mais curto para **t** em *G*

$\bar{c}_{ij} \leftarrow \rho_j - \rho_i + c_{ij}$, para todo (i, j) $\in$ *A*

A $\leftarrow$ { 1, ..., m } tal que $\forall$ i $\in$ { 1, ..., m }, $\bar{c}_i \leq \bar{c}_{i+1}$ se cauda(i) = cauda(i+1)

X $\leftarrow$ { p_1 }

h $\leftarrow$ 0

Enquanto (X $\neq \varnothing$) **e** (h $\leq$ k) **Fazer**

p $\leftarrow$ trajecto de X tal que $\bar{c}(p) \leq \bar{c}(q)$ para qualquer q $\in$ X

Se p não tem ciclos **Então**

h $\leftarrow$ h + 1

$p_h \leftarrow$ p

i $\leftarrow$ nó desvio de p

Repetir

$p_i \leftarrow$ subcaminho de p de **s** para **i**

b $\leftarrow$ arco de p cuja cauda é o nó **i**

Enquanto (i é a cauda de b+1) **e** (b+1 forma um ciclo com p_i) **Fazer**

b $\leftarrow$ b + 1

Se (i é a cauda de b+1) **Então**

j $\leftarrow$ cabeça de b + 1

$\bar{p}_j \leftarrow$ caminho mais curto de **j** para **t**

X $\leftarrow$ X $\cup$ { p_i $\lozenge$ [i, j] $\lozenge$ $\bar{p}_j$ } ($\lozenge$ = *concatenação*)

i $\leftarrow$ próximo nó de p

Até (p_i ser um ciclo) **Ou** (i = t)

Algoritmo 6. k caminhos mais curtos : baseado no cálculo de nós desvios (MPS).