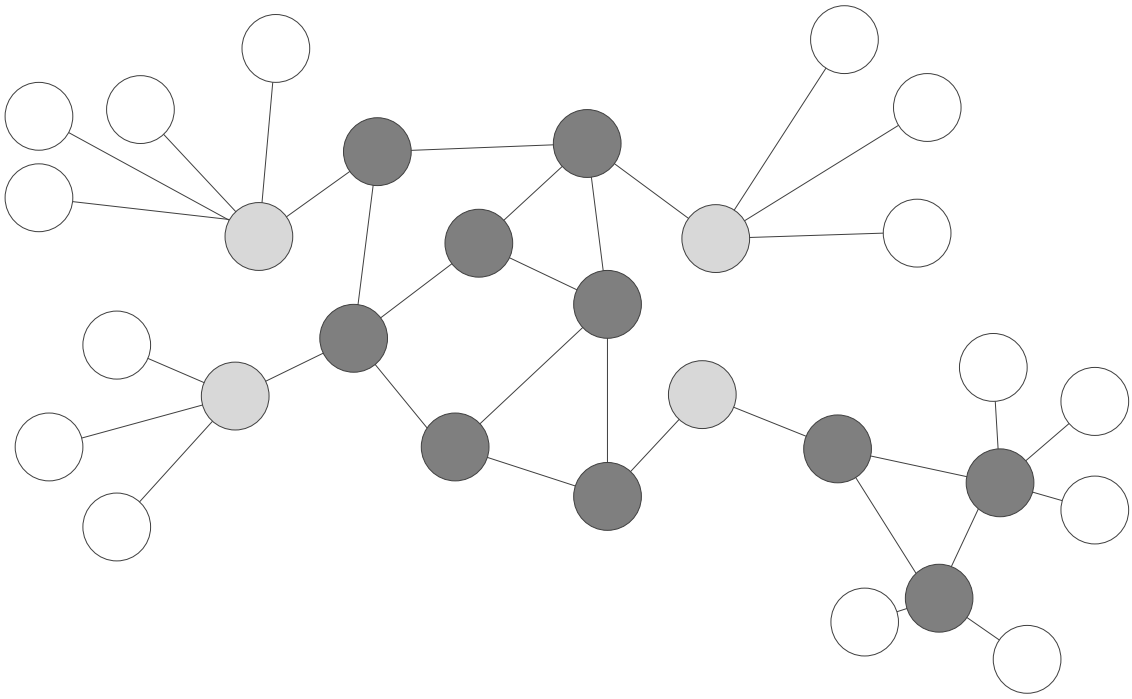
Universidade do Minho

Elementos de Engenharia de Sistemas
Módulo de Optimização de Redes

Introdução à Optimização
Introdução a Redes
Caminhos



Filipe Pereira e Alvelos

2016

v0.1

Índice

1
Introdução à Optimização

1.1 O problema geral de optimização

Está-se perante um problema de optimização quando se pretende escolher uma alternativa que é melhor do que todas as outras alternativas possíveis de acordo com um determinado objectivo. Um problema de optimização define-se quantitativamente, pela sua própria essência ou por conveniência.

Na terminologia da optimização, as alternativas designam-se por soluções e o grau de satisfação do objectivo traduz-se por uma função designada por função objectivo.

Formalmente, um problema de optimização para um determinado problema pode ser representado por

$$Min\ f(x)$$
$$sujeito\ a:$$
$$x \in X$$

em que x representa uma solução, X representa o conjunto das soluções possíveis e $f(x)$ é a função objectivo (a cada x faz corresponder o valor $f(x)$). Pretende-se identificar a (ou uma) solução x que minimiza a função objectivo – pretende-se o menor valor possível dessa função (o problema também poderia ser de maximização).

Uma solução possível x^* que tenha um valor dado pela função objectivo, $f(x^*)$, igual ou inferior em minimização (igual ou superior em maximização) a qualquer outra solução possível designa-se por solução óptima. Note-se que pode existir mais do que uma solução óptima (as quais são designadas por soluções óptimas alternativas). Tipicamente, pretende-se determinar apenas uma solução (idealmente óptima, embora, por vezes, tal possa não ser possível) para o problema de optimização.

A título exemplificativo, os problemas de optimização são relevantes na tomada de decisão em áreas como a localização de instalações ou serviços, o planeamento da produção, o escalonamento (*scheduling*) e sequenciamento da produção, o transporte e distribuição de bens, a definição de horários (*timetabling*), a gestão de projectos, a concepção e operação de redes telecomunicações.

A representação de problema de optimização geral apresentada acima admite os casos particularmente significativos da programação linear e inteira e da optimização combinatória que se discutem e exemplificam nas duas secções seguintes.

Na secção 1.4 introduz-se a extensão do problema do problema de optimização para vários objectivos – optimização multi-objectivo.

1.2 Programação linear e programação inteira

Se $f(x)$ for uma função linear em que x é um vector de variáveis de decisão (que traduzem as alternativas) e X puder ser representado através de equações e/ou inequações lineares, tem-se um modelo de programação linear ou programação (linear) inteira. No caso de todas as variáveis de decisão serem contínuas, o modelo é de programação linear. No caso de todas as variáveis de decisão serem binárias (0 ou 1) ou inteiras gerais (0, 1, 2, 3, ...) tem-se um modelo de programação (linear) inteira. O caso mais geral é quando existem variáveis de decisão contínuas e inteiras (binárias ou inteiras gerais) a que corresponde um modelo de programação inteira mista (mixed integer programming – MIP).

Implementações de métodos para programação linear (em particular, do algoritmo simplex) permitem resolver (i.e. obter uma solução óptima) modelos com um enorme número de variáveis de decisão e restrições (da ordem das centenas de milhar). A limitação ao tamanho dos modelos de programação linear que se conseguem resolver reside na memória computacional disponível.

Já relativamente à programação inteira não se pode afirmar o mesmo. Existe modelos com um número médio de restrições e variáveis de decisão (centenas) que não se conseguem resolver com os métodos mais avançados, mesmo após muito tempo de computação (semanas). Tal deve-se à inerente complexidade dos problemas de programação inteira, como se discutirá posteriormente. É de referir que, embora possa não ser possível obter uma solução óptima, tipicamente, a programação inteira permite obter soluções possíveis que podem ser de qualidade. Em programação inteira é ainda possível obter um limite superior para a distância do valor de uma solução possível ao valor óptimo.

Exemplo 1.1 Programação linear: actividades e recursos

Um uso frequente da programação linear é na determinação dos níveis a que devem ser realizadas diferentes actividades (produção, investimento, ...) tendo em conta que essas actividades usam recursos limitados comuns que não podem ser consumidos para além da sua disponibilidade.

Considerem-se os seguintes dados:

- n actividades;
- m recursos;
- p_j proveito unitário da actividade j, $j = 1, \dots n$;
- b_i disponibilidade do recurso i, $i = 1, \dots, m$;
- a_{ij} consumo unitário do recurso i pela actividade j, $i = 1, \dots, m$, $j = 1, \dots, n$.

As variáveis de decisão do modelo de programação linear são:

- x_j – nível da actividade j, $j = 1, \dots, n$.

O modelo de programação linear é:

$$Max \sum_{j=1}^{n} p_j x_j$$

$$sujeito\ a:$$

$$\sum_{j=1}^{n} a_{ij}\, x_j \leq b_i\,, i = 1, \ldots, m$$
$$x_j \geq 0, j = 1, \ldots, n.$$

Considere-se um caso concreto em que, numa fábrica, se pretende decidir as quantidades a produzir de três produtos (em Kg) de forma a maximizar o seu lucro. Cada um dos três produtos tem de ser processado em duas máquinas que estão disponíveis durante um período de tempo limitado (expresso em horas). Cada unidade de cada produto tem um tempo de processamento em cada uma das máquinas (expresso em horas). Na Tabela 1.1 são apresentados os dados relativos a este problema concreto.

	P1	P2	P3	Disponibilidade (h)
R1	13	12	15	200
R2	21	18	14	220
Proveito (€/Kg)	160	100	150	

Tabela 1.1. Dados de uma instância do problema de actividades e recursos.

Desta forma,

- $n = 3$ (produtos)
- $m = 2$ (recursos)
- $p_1 = 160, p_2 = 100, p_3 = 150$ (€/Kg)
- $b_1 = 200, b_2 = 220$ (h)
- $a_{11} = 13, a_{12} = 12, a_{13} = 15, a_{21} = 21, a_{22} = 18, a_{23} = 14\ (h/Kg)$

O modelo de programação linear é:

$$Max\ z = 160x_1 + 100 + 150x_3$$
$$sujeito\ a:$$
$$13x_1 + 12x_2 + 15x_3 \leq 200$$
$$21x_1 + 18x_2 + 14x_3 \leq 220$$
$$x_1, x_2, x_3 \geq 0$$

Através da utilização de *software* adequado (um *solver* de programação linear), é possível a obtenção de uma solução óptima para este problema, isto é, os valores de x_1, x_2 e x_3 que correspondem ao maior lucro possível.

■

Exemplo 1.2 Programação linear: transportes

Uma empresa pretende transportar um determinado produto de três locais onde está disponível (origens) para outros três locais onde é requerido (destinos). Nas origens (A, B e C) estão disponíveis 20, 30 e 40 unidades do produto, respectivamente. Nos destinos (1, 2 e 3) são necessárias 15, 25 e 50 unidades, respectivamente. Na Tabela 1.2 apresentam-se os custos de transportar uma unidade entre cada origem e cada destino. Pretende-se determinar as quantidades a enviar entre cada local origem e cada local destino de forma a minimizar o custo total de transporte.

	1	2	3
A	9	5	4
B	8	2	3
C	4	5	8

Tabela 1.2. Dados do problema de transportes.

Variáveis de decisão

$x_{ij} - quantidade\ a\ transportar\ da\ origem\ i\ para\ o\ destino\ j, i = 1,2,3, j = 1,2,3$

Modelo de Programação Linear

$$Min\ z = 9x_{11} + 5x_{12} + 4x_{13} + 8x_{21} + 2x_{22} + 3x_{23} + 4x_{31} + 5x_{32} + 8x_{33}$$

$sujeito\ a$:

$$x_{11} + x_{12} + x_{13} \leq 20$$
$$x_{21} + x_{22} + x_{23} \leq 30$$
$$x_{31} + x_{32} + x_{33} \leq 40$$
$$x_{11} + x_{21} + x_{31} = 15$$
$$x_{12} + x_{22} + x_{32} = 25$$
$$x_{13} + x_{23} + x_{33} = 50$$
$$x_{ij} \geq 0, i = 1,2,3, j = 1,2,3$$

■

O problema geral de transportes é modelado como apresentado de seguida.

Parâmetros

$n - número\ de\ origens$

$m - número\ de\ destinos$

$c_{ij} - custo\ unitário\ de\ transporte\ \ entre\ i\ e\ j, i = 1, \dots, n, j = 1, \dots, m$

$a_i - quantidade\ disponível\ na\ origem\ i, i = 1, \dots, n$

$b_j - quantidade\ requerida\ no\ destino\ j, j = 1, \dots, m$

Variáveis de decisão

$x_{ij} - quantidade\ a\ transportar\ de\ i\ para\ j, i = 1, \dots, n, j = 1, \dots m$

Modelo de Programação Linear

$$Min\ z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$sujeito\ a$:

$$\sum_{j=1}^{m} x_{ij} \leq a_i, i = 1, \dots, n$$

$$\sum_{i=1}^{n} x_{ij} = b_j, j = 1, \dots, m$$

$$x_{ij} \geq 0, i = 1, \dots, n, j = 1, \dots, m$$

As primeiras n restrições asseguram que, de cada origem, não saem mais unidades do que as lá existentes. As últimas m restrições asseguram que o número de unidades que chega a cada destino é o pretendido.

■

Exemplo 1.3 Programação inteira: mochila

O modelo da mochila pode ser visto como o modelo de programação inteira mais simples já que tem apenas uma restrição. Este modelo utiliza-se em situações em que se pretende seleccionar um subconjunto de itens, cada um caracterizado por um valor e um peso, de tal forma que o valor total (a soma dos valores dos itens seleccionados) seja o maior possível e o peso total (a soma dos pesos dos itens seleccionados) não ultrapasse um limite dado.

Considerem-se os seguintes dados:

- n itens;
- p_j valor do item j, $j = 1, \dots n$;
- w_j peso do item j, $j = 1, \dots, n$;
- b limite ao peso total.

As variáveis de decisão do modelo de programação inteira são

- $x_j = \begin{cases} 1, se\ item\ j\ é\ seleccionado \\ 0, caso\ contrário \end{cases}, j = 1, \dots, n.$

O modelo de programação inteira é:

$$Max \sum_{j=1}^{n} p_j x_j$$

$$sujeito\ a:$$

$$\sum_{j=1}^{n} w_j\, x_j \leq b$$

$$x_j \in \{0,1\}, j = 1, \dots, n.$$

A título exemplificativo do modelo da mochila num problema concreto, considere-se um conjunto de encomendas que se pretende entregar a diferentes clientes existindo apenas um veículo disponível com uma capacidade de 6 m^3. A cada encomenda estão associados o valor e o volume dados na Tabela 1.3. Pretende-se decidir as encomendas a entregar, já que não há capacidade suficiente para as entregar todas.

Encomenda	1	2	3	4	5	6	7	8	9
Valor (€)	123	134	125	112	140	124	133	122	150
Volume (m^3)	1.1	0.8	0.4	1.6	2.0	0.7	1.4	0.9	1.1

Tabela 1.3. Dados de uma instância do problema da mochila.

O modelo de programação inteira é

$$Max\ z = 123x_1 + 134x_2 + 125x_3 + 112x_4 + 140x_5 +$$
$$+124x_6 + 133x_7 + 122x_8 + 150x_9$$
$$sujeito\ a:$$
$$1.1x_1 + 0.8x_2 + 0.4x_3 + 1.6x_4 + 2.0x_5 + 0.7x_6 + 1.4x_7 +$$
$$+0.9x_8 + 1.1x_9 \leq 6$$
$$x_j \in \{0,1\}, j = 1, \dots, 9.$$

Através da utilização de *software* adequado (um *solver*), é possível a obtenção de uma solução óptima para este e problemas similares de muito maiores dimensões.

■

Exemplo 1.4 Programação inteira: afectação

Num determinado serviço de um hospital pretende-se fazer a escala de 10 enfermeiros para 10 turnos críticos. Cada enfermeiro deve trabalhar exactamente em um turno e em cada turno deve estar presente exactamente um enfermeiro. Na Tabela 1.4 são apresentadas as preferências de cada enfermeiro em relação a cada turno numa escala de 1 a 10 em que 1 corresponde à preferência máxima. Pretende-se determinar qual o turno que cada enfermeiro deve efectuar.

Turno \ Enfermeiro	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
1	1	1	1	3	5	3	5	1	1	2
2	3	2	2	7	9	9	6	2	3	1
3	2	10	3	1	3	2	7	4	2	3
4	8	4	4	4	6	7	1	5	4	5
5	9	5	5	2	4	6	2	9	5	4
6	4	3	6	5	7	5	3	10	6	8
7	5	6	7	8	10	4	8	6	10	7
8	7	7	8	9	1	10	9	8	9	6
9	6	8	9	10	2	8	10	7	7	10
10	10	9	10	6	8	1	4	3	8	9

Tabela 1.4. Dados do problema dos turnos dos enfermeiros.

Variáveis de decisão

$$x_{ij} = \begin{cases} 1, se\ enfermeiro\ i\ faz\ turno\ j \\ 0, caso\ contrário \end{cases}, i, \dots, 10; j = 1, \dots, 10$$

Modelo de PL

$$Min\ z = x_{11} + x_{12} + x_{13} + 3x_{14} + 5x_{15} + \cdots + 8x_{10,9} + 9x_{10,10}$$

$sujeito\ a$:

$$\sum_{j=1}^{10} x_{ij} = 1, i = 1, \dots, 10$$

$$\sum_{i=1}^{10} x_{ij} = 1, j = 1, \dots, 10$$

$$x_{ij} \in \{0,1\}, i = 1, \dots, 10; j = 1, \dots, 10$$

■

Em geral, num problema de afectação, existem n agentes para executar n tarefas. Cada agente executa exactamente uma tarefa e cada tarefa é executada exactamente por um agente. A tarefa j ser executada pelo agente i corresponde a um custo de c_{ij}. Pretende-se seleccionar a afectação entre agentes e tarefas de menor custo.

Variáveis de decisão

$$x_{ij} = \begin{cases} 1, se\ a\ afectação\ entre\ o\ agente\ i\ e\ a\ tarefa\ j\ é\ efectuada \\ 0, caso\ contrário \end{cases}, i = 1, \dots, n, j = 1, \dots, n$$

Modelo de Programação Inteira

$$Min\ z = \sum_{ij \in A} c_{ij} x_{ij}$$

$$sujeito\ a:$$

$$\sum_{j:ij \in A} x_{ij} = 1, i = 1, \dots, n$$

$$\sum_{i:ij \in A} x_{ij} = 1, j = 1, \dots, m$$

$$x_{ij} \in \{0,1\}, i = 1, \dots, n; j = 1, \dots, n$$

Neste problema as restrições de que forçam as variáveis a serem binárias podem ser substituídas por

$$0 \leq x_{ij} \leq 1, i = 1, \dots, n; j = 1, \dots, n$$

já que existe sempre uma solução óptima para o problema resultante em que as todas as variáveis tomam valores binários.

■

1.3 Optimização combinatória

Num problema de optimização combinatória pretende-se escolher o objecto que tem o menor valor de entre um conjunto finito de objectos. Os objectos em causa podem ser conjuntos de números inteiros, vectores de números inteiros, estruturas definidas em grafos e permutações, entre outros.

Seguindo a representação do problema de optimização geral, x corresponde a um objecto, X corresponde ao conjunto finito dos objectos possíveis e $f(x)$ é a função objectivo (ou função de avaliação).

Em geral, um problema de optimização combinatória pode ser modelado através de programação inteira. No entanto, em vários casos, existe uma clara vantagem em explorar a estrutura do problema incorporando conhecimento específico sobre o mesmo no método de resolução.

As abordagens mais frequentes para problemas práticos de optimização combinatória são heurísticas e dividem-se usualmente em heurísticas construtivas, heurísticas de pesquisa local e

as meta-heurísticas (os dois últimos tipos serão introduzidos posteriormente). Estes métodos não garantem a obtenção de uma solução óptima, sendo de salientar a sua eficiência e a sua flexibilidade na consideração de restrições e objectivos (a função objectivo pode ser vista como uma caixa negra que recebe uma solução e retorna um número, i.e., pode ser uma função linear, não linear, um algoritmo, ou mesmo um programa, e.g., de simulação).

Para alguns problemas de optimização combinatória, existem algoritmos específicos que exploram a estrutura do problema permitindo, tipicamente, obter soluções óptimas de forma muito eficiente. Esse grupo de problemas inclui o problema do caminho mais curto, o problema de afectação ou o problema da árvore de suporte de custo mínimo. Tipicamente, a extensão desses algoritmos muito eficientes para problemas com restrições adicionais não é trivial, sendo usados essencialmente aplicações directas ou em subproblemas.

Exemplo 1.5 Modelo de optimização combinatória: caixeiro viajante

O problema do caixeiro viajante ("travelling salesman problem" – TSP) consiste em, dado um número de cidades, n, juntamente com as distâncias entre todas elas, determinar a ordem pela qual devem ser visitadas todas as cidades (cada uma, uma e uma só vez) voltando àquela de que se partiu, percorrendo a menor distância possível.

Um método de optimização combinatória pode partir da representação de uma solução x como uma permutação das n cidades, X como o conjunto de todas as permutações de n cidades e $f(x)$ como a função que retorna o comprimento do circuito definido pela permutação x.

Exemplo 1.6 Heurística construtiva: mochila

Um método para um problema de optimização combinatória parte da definição da representação de uma solução. No caso do problema da mochila (introduzido no Exemplo 1.3), uma possível representação de uma solução é através de um conjunto de números inteiros (cada um correspondendo a um item).

Numa heurística construtiva, considera-se para solução inicial uma solução sem nenhum elemento. Em cada iteração é tentada a actualização da solução pela inclusão de um elemento que é seleccionado de acordo com uma regra. No problema da mochila, a regra poderá ser seleccionar o item com maior valor por unidade de peso (lembrando que o problema é de maximização). Para o caso do exemplo numérico anteriormente apresentado, os valores por unidade de peso são dados na Tabela 1.5.

Item	1	2	3	4	5	6	7	8	9
Valor	123	134	125	112	140	124	133	122	150
Peso	1.1	0.8	0.4	1.6	2	0.7	1.4	0.9	1.1
V/P	111.8	167.5	312.5	70.0	70.0	177.1	95.0	135.6	136.4

Tabela 1.5. Razões entre valor e peso no exemplo do problema da mochila.

Assim, na primeira iteração, o item 3 seria incluído no conjunto que define a solução. Na segunda iteração seria incluído o item 6. E nas seguintes os itens 2, 9, 8 e 1. O item seguinte, o 7, bem como os restantes (o 4 e o 5) não podem ser incluídos na solução por ultrapassarem a

capacidade (que é de 6 neste exemplo). A solução obtida pela heurística construtiva tem valor 778. Reforça-se que a solução dada por uma heurística não é necessariamente óptima.

■

1.4 Optimização multi-objectivo

Uma extensão do problema geral de optimização apresentado na secção 1.1 permite abordar problemas com mais do que um objectivo

$$Min\ f_1(x)$$
$$\ldots$$
$$Min\ f_m(x)$$
$$sujeito\ a:$$
$$x \in X$$

em que m é o número de objectivos.

De acordo com a estrutura do conjunto X, o problema pode ser visto como uma extensão para múltiplos objectivos do problema de programação linear, de programação inteira, ou de optimização combinatória.

A representação do problema de optimização multi-objectivo é também válida para problemas multi-atributo. Em problemas multi-atributo, as alternativas são conhecidas de forma explícita, portanto são conhecidas todas as soluções do conjunto X e os valores que cada uma toma em cada função objectivo. Embora este texto esteja centrado na optimização multi-objectivo, parte significativa é também válida para problemas multi-atributo.

Em optimização multi-objectivo (e também em multi-atributo) pretende-se obter uma solução preferida (por contraste com o conceito de solução óptima da optimização de objectivo único que não existe em optimização multi-objectivo). A solução preferida corresponde a um compromisso entre os valores das diferentes funções objectivo. Note-se que em problemas que justifiquem abordagens multi-objectivo, não faz sentido a existência de uma solução que é óptima para todos os objectivos quando estes são considerados individualmente. Tipicamente, os objectivos são conflituosos (uma melhoria num objectivo implica a deterioração de outro(s)).

Dois conceitos fundamentais em optimização multi-objectivo (e multi-critério) são o de solução eficiente e solução dominada. Uma solução é eficiente (ou óptima de Pareto) quando não existe nenhuma outra solução que seja melhor do que ela em, pelo menos, um dos objectivos. O conjunto das soluções eficientes designa-se por fronteira eficiente (ou fronteira de Pareto). Uma solução diz-se dominada quando existe uma solução que é melhor que do que ela em, pelo menos, um objectivo.

A solução preferida é necessariamente uma solução eficiente. A escolha de uma qualquer solução eficiente pode ser justificada racionalmente. Assim, o processo de determinação da solução preferida (necessariamente de entre as soluções eficientes) tem de incluir a perspectiva do agente de decisão. De facto, apenas a preferência de um agente de decisão pode determinar a escolha de uma solução que é melhor do que outra(s) num objectivo mas pior noutro(s).

Os métodos para problemas de optimização multi-objectivo podem ter como propósito a obtenção da solução preferida ou a geração de um conjunto de soluções eficientes que serão posteriormente analisados pelo agente de decisão.

Exemplo 1.7 Optimização multi-objectivo: mochila

Considere-se o problema da mochila (introduzido no Exemplo 1.3) multi-objectivo com o objectivo adicional de maximizar a soma dos índices de urgência das encomenda (índice de 1 a 5 em que 5 corresponde à urgência máxima).

Encomenda	A	B	C	D	E	F	G	H	I
Valor (€)	123	134	125	112	140	124	133	122	150
Urgência (1-5)	2	1	4	5	5	3	1	4	4
Volume (m^3)	1.1	0.8	0.4	1.6	2.0	0.7	1.4	0.9	1.1

Tabela 1.6. Dados de uma instância do problema da mochila multi-objectivo.

Um modelo de optimização multi-objectivo é:

$$Max\ z_1 = 123x_1 + 134x_2 + 125x_3 + 112x_4 + 140x_5 +$$
$$+124x_6 + 133x_7 + 122x_8 + 150x_9$$
$$Max\ z = 2x_1 + x_2 + 4x_3 + 5x_4 + 5x_5 +$$
$$+3x_6 + 1x_7 + 4x_8 + 4x_9$$
$$sujeito\ a:$$
$$1.1x_1 + 0.8x_2 + 0.4x_3 + 1.6x_4 + 2.0x_5 + 0.7x_6 + 1.4x_7 +$$
$$+0.9x_8 + 1.1x_9 \leq 6$$
$$x_j \in \{0,1\}, j = 1, \dots, 9.$$

Considere-se agora o problema seleccionar apenas uma encomenda. As soluções possíveis e os seus valores em ambos os objectivos estão representadas na Figura 1.1.

As soluções E e I são as duas únicas soluções eficientes. Dependendo do método usado e das preferências do agente de decisão, uma delas será a solução preferida.

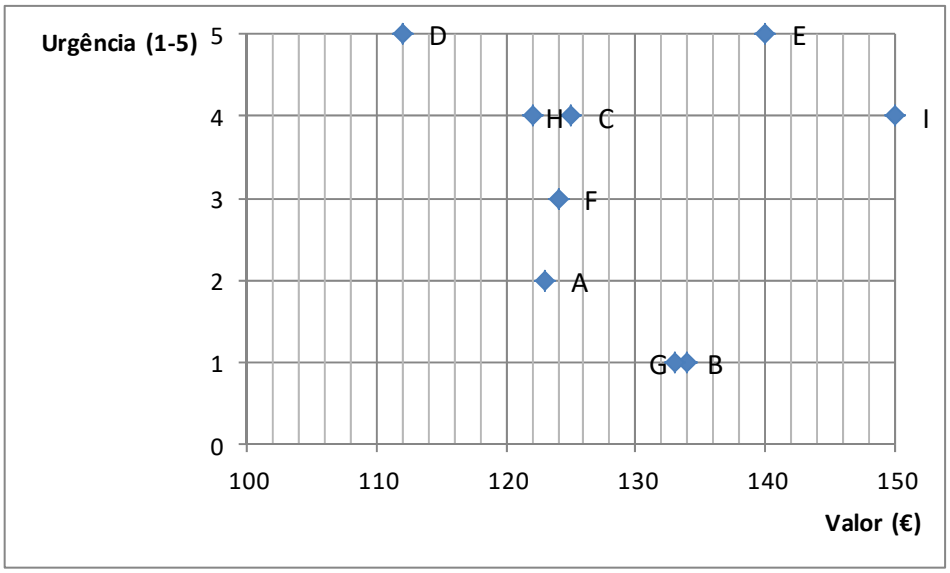


Figura 1.1. Representação no espaço dos objectivos do Exemplo 1.7.

■

1.5 *Eficiência de algoritmos e complexidade computacional

Um aspecto relevante em optimização é, antes de atacar um problema, ter a percepção de quão difícil o problema é. Há problemas para os quais a programação linear ou métodos específicos garantem a obtenção de uma solução óptima com pequenos tempos computacionais para instâncias virtualmente de qualquer dimensão. Para outros problemas, a obtenção de soluções óptimas em instâncias de alguma dimensão é virtualmente impossível.

Estes conceitos estão relacionados com o estudo da eficiência de algoritmos e da complexidade de problemas, temas que agora se introduzem.

1.5.1 Eficiência de algoritmos

Um algoritmo é eficiente quando um ligeiro aumento no tamanho da instância (medida pelo número de dígitos de entrada do algoritmo ou por outra uma medida representativa – por exemplo o número de cidades no problema do caixeiro viajante) a resolver provoca um ligeiro aumento no tempo de execução do algoritmo. Pelo contrário, um algoritmo não é eficiente se um pequeno aumento no tamanho da instância implica um grande aumento no tempo de execução do algoritmo.

Em optimização as duas formas mais frequentes de avaliar a eficiência de algoritmos são a análise do pior caso (teórica) e a aplicação do algoritmo a um conjunto de instâncias do problema (experimental).

A análise do pior caso baseia-se na consideração de que as operações elementares de um algoritmo (somas, comparações, atribuições, …) demoram uma unidade de tempo (o que torna a análise independente de linguagens de programação e de computadores).

Exemplo 1.8 Análise de pior caso: "Bubble sort"

O algoritmo "bubble sort" ordena uma lista de n elementos.

```
// v(i) tem o elemento na posição i
Fazer
     haTroca=false
     Para i=1 até n-1
          Se v(i)>v(i+1) então
               troca v(i) com v(i+1)
               haTroca=true
Enquanto haTroca==true
```

Um exemplo da execução do algoritmo para a ordenação de 10 elementos é dado na Figura 1.2.

iter	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
pos1	91	39	39	39	39	39	39	39	10	10
pos2	39	91	41	41	41	41	41	10	39	21
pos3	98	41	71	58	58	41	10	21	21	39
pos4	41	71	58	71	41	10	21	41	41	41
pos5	71	58	91	41	10	21	41	41	41	41
pos6	58	96	41	10	21	58	58	58	58	58
pos7	96	41	10	21	71	71	71	71	71	71
pos8	41	10	21	91	91	91	91	91	91	91
pos9	10	21	96	96	96	96	96	96	96	96
pos10	21	98	98	98	98	98	98	98	98	98

Figura 1.2. Exemplo de execução do "bubble sort".

O algoritmo tem dois ciclos: o ciclo exterior (`Fazer ... Enquanto`) e o ciclo interior (`Para i=1 até n-1`).

O melhor caso ocorre quando a lista já está ordenada: o ciclo exterior é executado apenas uma vez e no ciclo interior são efectuadas $n-1$ comparações. Assim, o número de operações elementares efectuadas é de $n-1$.

No pior caso, o ciclo exterior é executado $n-1$ vezes e em cada uma delas são efectuadas as $n-1$ comparações do ciclo interior. Assim, o número de operações elementares efectuadas é $(n-1)^2$.

É de notar que embora o "bubble sort" seja um algoritmo eficiente, é possível melhorar a sua eficiência (o ciclo interior pode ser encurtado já que em cada iteração há um elemento que fica na sua posição final).

■

Exemplo 1.9 Análise de pior caso: caixeiro viajante

Teoricamente, qualquer problema de optimização combinatória pode ser resolvido por enumeração completa (também designada por pesquisa exaustiva). No caso do problema do caixeiro viajante, tal corresponde a calcular a distância associada a cada uma das permutações e escolher uma permutação a que corresponda a menor distância. O número de permutações de n elementos é $n!$, sendo este o número de operações elementares do algoritmo de enumeração completa[1]. A Tabela 1.7. ilustra a impossibilidade desta abordagem mesmo para instâncias pequenas.

n (número de cidades)	$n!$ (número de soluções)	Observações
10	3628800	
30	2.65×10^{32}	Testando um bilião de alternativas por segundo (um computador muito bom!), o tempo total seria mais de oito milénios Estima-se que a idade do universo seja 4.4×10^{17} segundos
100	9.3×10^{157}	Estima-se que o número de átomos no universo esteja entre 10^{78} e 10^{82}

Tabela 1.7. Número de soluções para instâncias do caixeiro viajante com diferentes dimensões.

É de notar que existem algoritmos para o problema do caixeiro viajante bem mais eficientes do que a enumeração completa. Esses algoritmos combinam programação inteira com heurísticas e métodos específicos para o problema.

Apenas para instâncias muito pequenas, é razoável a utilização de algoritmos de enumeração completa. Estes algoritmos não são eficientes.

■

É de notar que pode haver problemas para os quais um algoritmo é mais eficiente do que outro na prática, embora menos eficiente na análise de pior caso. É o caso dos algoritmos simplex e do elipsóide para programação linear. Embora o último seja mais eficiente na análise de pior caso, o simplex é muito mais eficiente na prática, estando implementado em virtualmente todo o software para programação linear.

1.5.2 Complexidade computacional

A complexidade computacional de um algoritmo é $O(f(n))$ se existir uma constante c tal que o tempo de execução para todas as instâncias de tamanho n é limitado por $cf(n)$.

[1] Se o problema for simétrico (i.e., se a distância entre i e j for igual à distância entre j e i), o número reduz-se a metade o que, dado que a sua ordem de grandeza permanece enorme, não altera as conclusões que aqui se extraem.

Por exemplo, a complexidade computacional do algoritmo "bubble sort" é $O(n^2)$ e o de enumeração completa para o problema do caixeiro viajante é $O(n!)$. A notação agora introduzida, notação do O-maiúsculo, usa as seguintes simplificações:

- se $f(n)$ é uma soma de vários termos, apenas aquele com maior crescimento é mantido;
- se $f(n)$ é um produto de vários factores, as constantes são omitidas.

Por exemplo, se a eficiência de um algoritmo é dada pela função $f(n) = 2n^3 + 4n^2 - 3n$, a sua complexidade computacional é $O(f(n)) = O(n^3)$.

Se a complexidade computacional de um algoritmo é polinomial, o algoritmo diz-se polinomial e é eficiente. Caso contrário, o algoritmo diz-se exponencial e não é eficiente.

Exemplos de crescimento de diferentes funções são dados na Tabela 1.8.

n	10	100	1000
$n\ log_2(n)$	33.2	664.4	9965.8
n^3	1000	1000000	1000000000
$10^6 n^8$	1E+14	1E+22	1E+30
2^n	1024	1.3E+30	1.072E+301
$n^{log_2(n)}$	2098.6	1.9E+13	7.895E+29
$n!$	113227	9E+157	4E+2567

Tabela 1.8. Exemplos do crescimento de funções.

Esta classificação da complexidade computacional, permite caracterizar os problemas de optimização em dois grupos: fáceis e difíceis. Se existe um algoritmo eficiente para o problema de optimização, o problema é fácil (formalmente pertence a uma classe de problemas designada por P). Caso contrário, é difícil (formalmente é um problema NP-difícil).

A existência de um algoritmo polinomial para os problemas NP-difíceis está em aberto (sendo generalizada a suspeição da não existência).

Exemplos de problemas fáceis: caminho mais curto, árvore de suporte de custo mínimo, afectação, fluxo de custo mínimo e programação linear. Exemplos de problemas difíceis: caixeiro viajante, encaminhamento de veículos, localização, programação inteira.

1.6 Exercícios

Exercício 1.1 Produção

Uma fábrica produz cinco produtos (A, B, C, D e E) com base em três processos: polimento, perfuração e montagem. O lucro associado a cada um dos produtos é dado na tabela.

Produto	A	B	C	D	E
Lucro (€/unidade)	550	600	350	400	200

A produção de cada unidade exige um determinado tempo em cada processo, valores que são dados na tabela (em horas).

	A	B	C	D	E
Polimento	12	20	−	25	15
Perfuração	10	8	16	−	−

Adicionalmente, a montagem de cada unidade consome 20 horas.homem. As máquinas de polir e as máquinas de perfurar estão disponíveis 400 e 300 horas por semana, respectivamente. A fábrica tem oito colaboradores dedicados à operação de montagem e cada um deles trabalha oito horas por dia, cinco dias por semana. Apresente um modelo de programação linear que permita determinar quais as quantidades a produzir por semana de forma a maximizar o lucro total.

Exercício 1.2 Política de vacinação

Os responsáveis pela política de vacinas por uma determinada região deparam-se com o problema de minimizar o valor monetário dispendido com a aquisição e transporte das vacinas para os Centros de Saúde onde serão administradas. O número de vacinas necessário em cada um dos 4 Centros de Saúde da referida região é de 20 000, 50 000, 30 000 e 40 000. Existem três empresas farmacêuticas capazes de fornecer vacinas no prazo estipulado, tendo cada uma feito uma proposta.

A empresa A coloca em cada Centro de Saúde as vacinas que forem precisas a um preço de 0.5€ por vacina.

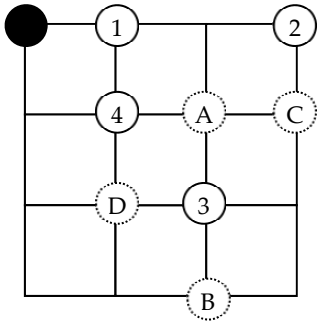
A empresa B vende cada vacina por 0.2€ mas não se responsabiliza pelo seu transporte para os Centros de Saúde. Esta empresa não assegura mais de 100 000 vacinas. Estima-se o custo de transporte por vacina das instalações da empresa B para cada um dos Centros de Saúde em 0.15€, 0.2€, 0.35€ e 0.4€ (pela ordem em que foram inicialmente referenciados).

A empresa C garante a entrega de, no máximo, 80 000 vacinas apenas aos Centros de Saúde 1, 2 e 3 e pelos preços de 0.2€, 0.4€ e 0.3€.

Apresente um modelo de programação linear que permita determinar a política de vacinação com menor custo.

Exercício 1.3 Armazenamento e recolha

Num determinado armazém, pretendem-se recolher quatro objectos e guardar outros quatro objectos. No esquema seguinte são indicadas as posições dos quatros objectos que se pretende recolher (1, 2, 3 e 4) e as posições onde se pretende guardar os outros quatro objectos (A, B, C e D).



O círculo a negro representa a posição onde se encontram inicialmente o veículo responsável pelas operações e os objectos A, B, C e D. Os objectos 1, 2, 3 e 4 têm de ser recolhidos para esse mesmo local.

A grelha do esquema representa percursos que o veículo pode seguir. Considere que a unidade de distância é o lado do quadrado da grelha. O veículo apenas pode carregar um objecto de cada vez. Por exemplo, uma viagem poderá consistir em sair do local inicial com o objecto A, guardá-lo, recolher o objecto 4 e voltar à posição inicial. A distância percorrida é 6.

Apresente um modelo de programação inteira que lhe permita determinar como devem ser guardados e recolhidos os objectos de forma a que o veículo percorra a menor distância possível.

Exercício 1.4 ONG

A Direcção de uma determinada ONG (Organização Não Governamental) pretende decidir para onde deve enviar as suas equipas de ajuda humanitária. Numa primeira análise foram identificados 10 destinos possíveis correspondendo a 10 diferentes regiões. Para cada um desses destinos foi estimado o benefício humanitário (numa escala de 0 a 100) da presença de uma equipa da referida ONG e o custo a ela associado. Esses valores são dados na tabela abaixo.

O orçamento da referida ONG para o período em questão é de 250 U.M.. Considera-se que está fora de questão o envio de mais de uma equipa para uma mesma região e que o número de equipas disponíveis não é uma restrição.

Região	1	2	3	4	5	6	7	8	9	10
Benefício (0-100)	90	60	80	50	20	40	95	45	15	30
Custo (U.M.)	100	50	80	40	25	50	80	45	10	20

a) Apresente um modelo de programação inteira que maximize o benefício total (soma dos benefícios de cada região).

b) Obtenha uma solução com uma heurística construtiva. A solução que obteve é óptima?

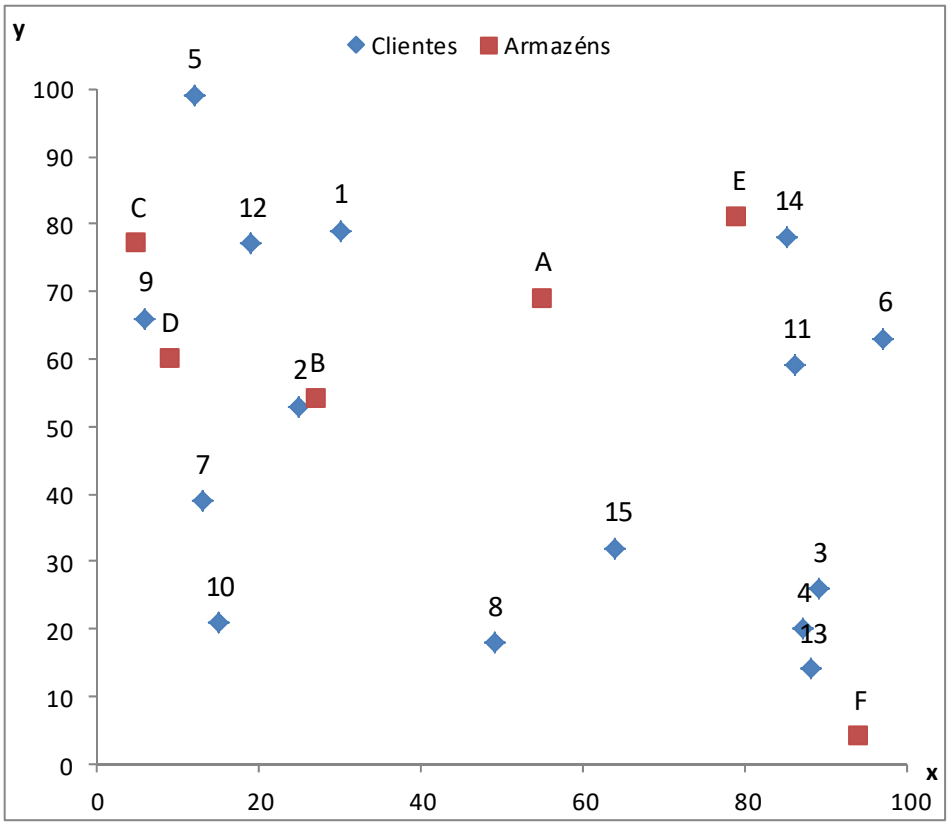
c) Considere agora dois objectivos: maximizar o benefício e minimizar o custo e que que apenas uma equipa pode ser enviada. Represente o problema no espaço dos objectivos

Exercício 1.5 Localização

Considere-se o problema de seleccionar a localização de um armazém para servir um conjunto de clientes. As potenciais localizações do armazém e a localização dos clientes são dados na tabela e representados na figura.

Clientes	X	Y
1	30	79
2	25	53
3	89	26
4	87	20
5	12	99
6	97	63
7	13	39
8	49	18
9	6	66
10	15	21
11	86	59
12	19	77
13	88	14
14	85	78
15	64	32

Potenciais armazéns	X	Y
A	55	69
B	27	54
C	5	77
D	9	60
E	79	81
F	94	4



A abertura de um armazém específico corresponde a uma solução para o problema. Consideram-se dois critérios (ambos de minimização): o custo de instalação do armazém e a distância aos clientes. Cada critério é caracterizado por um atributo: valor monetário (em €) e soma das distâncias de todos os clientes ao armazém (em Km).

Os valores das diferentes soluções para cada um dos atributos são dados na Tabela 1.9.

	Distância (Km)	Custo (€)
A	687	900
B	667	1200
C	906	1400
D	801	2100
E	841	800
F	1019	400

Tabela 1.9. Valor em cada atributo de cada solução do exemplo da localização multi-atributo.

a) Apresente o conjunto das soluções eficientes.

b) Considere agora que se pode abrir (até) dois armazéns. Apresente o conjunto das soluções eficientes.

1.7 Resultados de aprendizagem

- Formular modelos de programação linear de actividades e recursos, e transportes.
- Formular modelos de programação inteira de mochila e afectação.
- Aplicar uma heurística construtiva para o problema da mochila.
- Identificar soluções dominadas e eficientes em problemas multi-atributo.
- Identificar o tipo de modelo entre programação linear, programação inteira, optimização combinatória e optimização multi-objectivo.

2
Introdução a Redes

2.1 Motivação

Uma rede é uma representação de um sistema (real ou idealizado) e consiste num conjunto de vértices (nodos ou nós) e um conjunto de pares de vértices designados por arestas (arcos ou ligações), juntamente com informação associada aos vértices ou às arestas.

A importância das redes e do seu estudo provém do grande número de sistemas das mais diversas áreas de actividade humana e da natureza poderem ser representados através de redes. Tal justifica-se, em grande parte, por as redes permitirem modelar sistemas em que a comunicação (entendida em sentido lato) eficiente entre os seus componentes não é toda feita directamente entre pares de componentes.

Note-se que são raros os sistemas complexos em que todos os pares de componentes comunicam (ou em que se pretende que comuniquem) directamente. Um exemplo, de entre muitos, de um sistema em que tal não acontece é o transporte aéreo. O transporte aéreo pode ser modelado através de uma rede em que os nodos são os aeroportos e os arcos as ligações aéreas (entre aeroportos). Claramente, não é vantajoso existirem ligações aéreas directas entre aeroportos cuja procura de viagens é muito reduzida. Tal implicaria a existência de voos com taxas de ocupação muito baixas. Por causa da estrutura em rede, tal não impede que seja possível viajar entre eles (fazendo escala noutros aeroportos).

A representação em rede de um sistema pode ser mais ou menos directa. Por exemplo, a representação em rede de uma rede de computadores ou de uma rede rodoviária são directas. Já a representação em rede de sistemas de produção, tipicamente, envolve um maior nível de abstracção.

Em qualquer dos casos, uma representação em rede do sistema em estudo, em geral, permite:

- a percepção do sistema de forma intuitiva, já que visual;
- a modelação do sistema (e em particular das ligações entre os seus componentes) com conceitos rigorosos que formam um corpo teórico coerente;
- a utilização de ferramentas (e.g. algoritmos) independentes do sistema para definir a estrutura, obter medidas ou dimensionar parâmetros relevantes do sistema.

A utilidade das redes prende-se ainda com o potencial da associação de valores aos vértices ou arestas, como por exemplo:

- Capacidade de uma aresta (por exemplo, a largura de banda do cabo que estabelece a ligação entre os dois computadores representados pela aresta);
- Custo de atravessar uma aresta (por exemplo, a distância entre os dois locais representados pelos vértices unidos pela aresta);

- Procura de um vértice (por exemplo, número de unidades de um determinado produto a serem entregues ao cliente representado pelo vértice).

2.2 Definições

2.2.1 Definições básicas

Na base de uma representação em rede, está o conceito matemático de grafo. Uma rede pode ser vista como um grafo ao qual se adiciona informação. Formalmente, um grafo $G = (N, A)$ em que N é o conjunto de vértices e A é o conjunto de arestas, $a = \{i, j\}, i \in N, j \in N, \forall a \in A$. Na Figura 2.1 é dada uma visualização de um grafo em que $N = \{1,2,3,4,5,6,7\}$ e $A = \{\{1,2\}, \{1,3\}, \{2,4\}, \{3,2\}, \{3,4\}, \{4,5\}, \{5,2\}, \{5,6\}, \{6,4\}, \{6,7\}, \{7,3\}\}$.

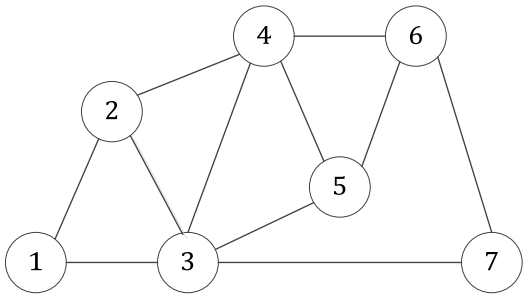


Figura 2.1. Um grafo com 7 vértices e 11 arestas.

Quando uma rede tem apenas um valor associado a cada arco também se designa por grafo com pesos.

Excepto quando afirmado algo em contrário, os grafos e redes estudados neste texto não têm arestas com origem e destino no mesmo nodo (não existem anéis) e entre dois vértices existe, no máximo, uma aresta (não existem arestas múltiplas).

As arestas que têm o vértice i como extremo são designadas por arestas incidentes em i.

O **grau** do vértice i é o número de arestas incidentes em i. Dois vértices unidos por uma aresta são designados por adjacentes.

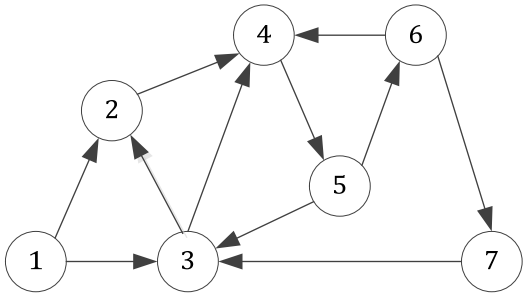
Um **caminho** (elementar ou simples) é uma sequência de vértices distintos e das arestas que os unem.

Um **circuito** (ou ciclo) (elementar ou simples) é uma sequência de vértices distintos (com excepção do primeiro e do último) e das arestas que os unem em que a última aresta liga o último vértice ao primeiro.

É usual omitirem-se os vértices ou as arestas na representação de caminhos e circuitos.

A título exemplificativo, no grafo da Figura 2.1, as arestas incidentes em 4 são as {2,4}, {3,4}, {4,5} e {6,4}; o vértice 6 tem grau 3; os vértices 3 e 7 são adjacentes; $\{1,2\} - \{2,4\} - \{4,5\}$ é um caminho entre 1 e 5; $3 - 4 - 5 - 3$ é um circuito.

O grafo pode ser não orientado ou **orientado**. Num grafo não orientado, $\{i,j\} = (i,j) = (j,i)$. Num grafo orientado, (i,j) não é o mesmo que (j,i). Usualmente, num grafo não orientado as ligações designam-se por arestas e num grafo orientado as ligações designam-se por arcos. Um exemplo de grafo orientado é dado na Figura 2.2.



$N = \{1,2,3,4,5,6,7\}$
$A = \{(1,2), (1,3), (2,4), (3,2), (3,4), (4,5), (5,2), (5,6), (6,4), (6,7), (7,3)\}$

Figura 2.2. Um grafo orientado.

Num grafo orientado, um caminho / circuito orientado é um caminho / circuito em que nenhum arco é percorrido em sentido contrário à sua orientação (usualmente, dado que se pode inferir pelo contexto, usa-se caminho / circuito com o significado de caminho / circuito orientado).

Num grafo orientado o grau de um nodo pode ser separado em grau de entrada (número de arcos que entram no nodo) e grau de saída (número de arcos que saem do nodo).

2.2.2 Definições adicionais

Providenciam-se agora definições adicionais, relativamente às dadas na subsecção 2.2.

Circuitos Hamiltonianos e Eulerianos

Um **circuito Hamiltoniano** de um grafo é um circuito que inclui todos os seus vértices.

Um **circuito Euleriano** (de Euler, 1707-1783) é um circuito que inclui todos os arcos uma e uma só vez (nodos podem ser repetidos). A primeira abordagem a um problema através de grafos deve-se a Euler. O problema em causa é o problema das pontes de Könisberg que consistia em determinar se é possível, começando numa zona qualquer, atravessar todas as pontes da cidade de Könisberg (Figura 2.3) uma só vez e regressar ao ponto inicial.

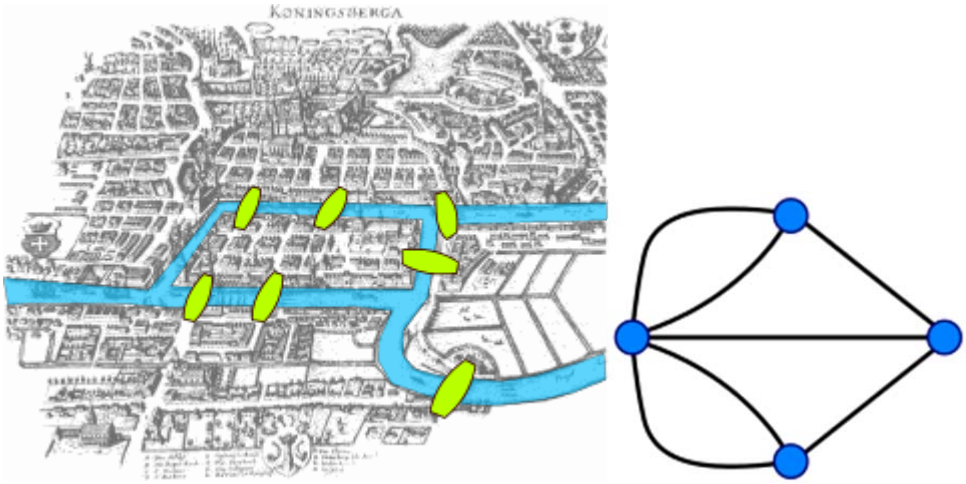


Figura 2.3. Ilustração da cidade de Könisberg e grafo do Problema das pontes Könisberg: ilustração da cidade e grafo correspondente.[2]

Euler demonstrou o seguinte teorema (teorema de Euler): um grafo não orientado tem um circuito que passa por todos os arcos uma e uma só vez (circuito Euleriano) se e só se todos os vértices tiverem grau par.

Assim, não é possível atravessar todas as pontes da cidade de Könisberg uma só vez e regressar ao ponto inicial.

Grafos conexos, bipartidos e árvores

Um grafo é **conexo** se existe pelo menos um caminho entre qualquer par de nodos.

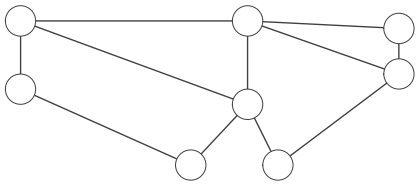


Figura 2.4. Grafo conexo.

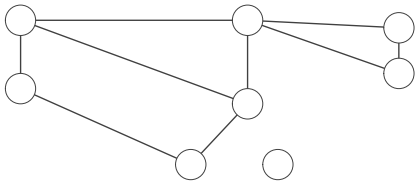


Figura 2.5. Grafo não conexo.

[2] https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

Um grafo é **bipartido** se o seu conjunto de vértices pode ser dividido em dois subconjuntos de tal forma que cada dois vértices adjacentes fazem parte de subconjuntos diferentes. Os subconjuntos referidos designam-se por partições.

Como é claro da representação da direita, o grafo da Figura 2.6 ser bipartido resulta na sua partição nos subconjuntos $N_1 = \{1,5,4,8\}$ e $N_2 = \{2,3,6,7\}$.

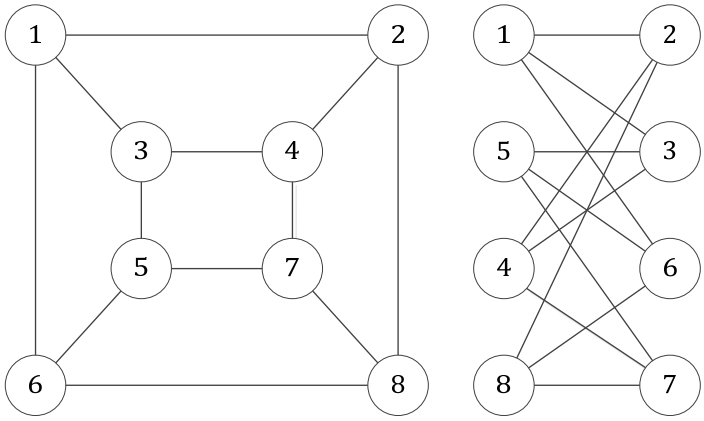


Figura 2.6. Um grafo bipartido representado de duas formas.

Uma **árvore** é um grafo conexo sem circuitos.

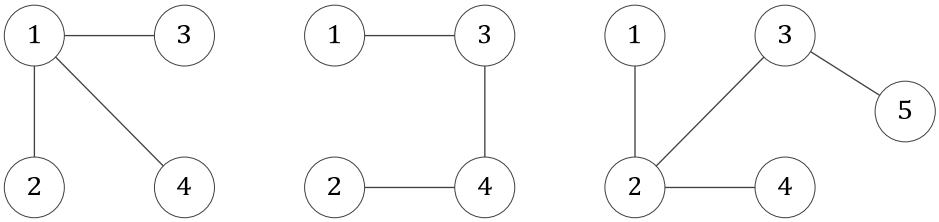


Figura 2.7. Três árvores.

Dado um grafo conexo G, designa-se por árvore de suporte todo o subgrafo de G que seja uma árvore e inclua todos os vértices de G.

Um subgrafo de um grafo conexo G com n vértices é uma árvore de suporte se se verificar uma das seguintes condições equivalentes:

- ter *n–1* arestas e ser conexo;
- ter *n–1* arestas e não ter circuitos;
- existir um único caminho entre qualquer par de vértices de G;
- não ter circuitos, mas a adição de uma aresta resultar num circuito.

Uma **floresta** é um grafo sem circuitos (ou, de forma equivalente, uma floresta é um conjunto de árvores).

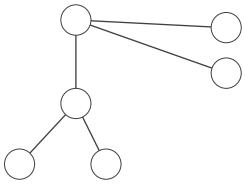


Figura 2.8. Um subgrafo do grafo da Figura 2.4 que é uma árvore mas não de suporte.

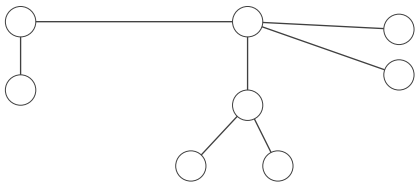


Figura 2.9. Um subgrafo do grafo da Figura 2.4 que é uma árvore de suporte.

Corte

Num grafo $G = (N, A)$, um **corte** é uma partição do conjunto de nodos N em dois subconjuntos, S e $\bar{S} = N - S$. Cada corte define um conjunto de arcos formado pelo arcos que têm uma extremidade em S e outra extremidade em $\bar{S}$.

Na Figura 2.10 representa-se o corte associado a $S = \{1,2\}$. O conjunto de arcos associado a este corte é $\{(1,3), (2,4), (3,2)\}$.

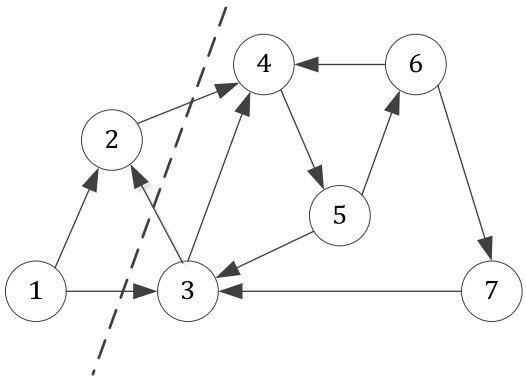


Figura 2.10. Um corte.

2.3 Representações em computador

Para lidar com uma rede computacionalmente, é necessário ter uma sua representação em estruturas de dados adequadas.

Para exemplificar as representações mais usuais, considera-se uma rede orientada com n nodos e m arcos e um parâmetro associado aos arcos a uma instância dada na Figura 2.11.

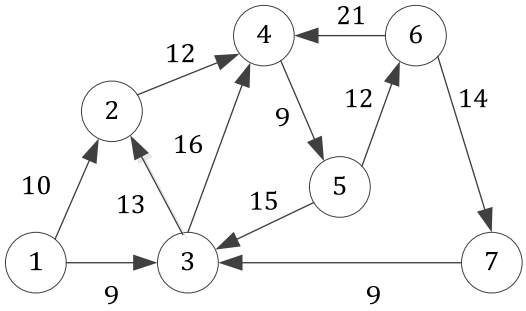


Figura 2.11. Rede do exemplo.

2.3.1 Lista de arcos

A representação mais simples de uma rede num computador é através de um vector de arestas / arcos aos quais se podem associar outros vectores com parâmetros como ilustrado na Tabela 2.1.

Índice	origem arco	destino arco	Parâmetro
1	1	2	10
2	2	3	9
3	2	4	12
4	3	2	13
5	3	4	16
6	4	5	9
7	5	3	15
8	5	6	12
9	6	4	21
10	6	7	14
11	7	3	9

Tabela 2.1. Exemplo de representação de uma rede através de uma lista de arcos.

Embora a utilização de espaço seja eficiente (todos os elementos têm informação, embora nalguns casos repetida – ver estrela de sucessores), esta representação tem a grande desvantagem das operações básicas como, por exemplo, testar a existência de um arco ou percorrer todos os arcos que saem de um dado nodo não serem feitas de forma eficiente (em ambos os exemplos, requerem percorrer todos os arcos da rede).

2.3.2 Matriz de adjacência (nodo-nodo)

Na representação de uma rede através da matriz adjacência, é definida uma matriz em que cada linha está associada a um nodo e cada uma coluna está também associada a um nodo. O elemento *ij* é 1 se arco com origem em *i* e destino *j* existe e é 0 caso contrário. Os parâmetros são guardados em matrizes similares. As Tabela 2.2 e Tabela 2.3 exemplificam esta a representação da matriz de adjacência.

	1	2	3	4	5	6	7
1	–	1	1				
2		–		1			
3		1	–	1			
4				–	1		
5			1		–	1	
6				1		–	1
7			1				–

Tabela 2.2. Exemplo de representação de uma rede através de uma matriz de adjacência.

	1	2	3	4	5	6	7
1	–	10	9				
2		–		12			
3		13	–	16			
4				–	9		
5			15		–	12	
6				21		–	14
7			9				–

Tabela 2.3. Exemplos de parâmetros na representação de uma rede através de uma matriz de adjacência.

A matriz de adjacência tem n^2 elementos dos quais m são diferentes de zero o que significa um grande desperdício de memória, em particular na representação de redes esparsas – com poucos arcos.

As grandes vantagens desta representação são a simplicidade de algumas operações (por exemplo, para percorrer arcos que entram ou saem de um nodo, basta percorrer a coluna ou linha, respectivamente, desse nodo) e a eficiência de outras (por exemplo, testar a existência de um arco ou ler um parametro).

2.3.3 Matriz de incidência (nodo-arco)

Na representação de uma rede através da matriz incidência, é definida uma linha para cada nodo e uma coluna para cada arco. Em cada coluna, um e um só elemento tem valor +1 e um e um só elemento tem valor –1. O elemento ij é 1 se arco da coluna j tem origem no nodo i, é –1 se arco da coluna j tem destino no nodo i, é 0 caso contrário. Os parâmetros são guardados em vectores de dimensão m com mesma ordenação que as colunas. As Tabela 2.4 e Tabela 2.5 exemplificam esta a representação da matriz de adjacência.

A matriz de incidência tem nm elementos dos quais $2m$ são diferentes de zero. O número de +1 (–1) de uma linha (coluna) corresponde ao número de arcos que saem (entram) do (no) nodo.

Esta representação é ainda menos eficiente em termos de espaço do que a matriz de adjacência. No entanto, a sua estrutura torna particularmente simples a construção de modelos de programação linear e inteira para problemas de redes.

	12	23	24	32	34	45	53	56	64	67	73
1	1										
2	-1	1	1	-1							
3		-1		1	1		-1				-1
4			-1		-1	1			-1		
5						-1	1	1			
6								-1	1	1	
7										–1	1

Tabela 2.4. Exemplos da representação de uma rede através de uma matriz de incidência.

	12	23	24	32	34	45	53	56	64	67	73
	10	9	12	13	16	9	15	12	21	14	9

Tabela 2.5. Exemplos de parâmetros na representação de uma rede através de uma matriz de incidência.

2.3.4 Estrela de sucessores

O primeiro passo para obter uma representação por estrela de sucessores, é ordenar os arcos por origem, como mostrado na Tabela 2.6.

Índice	origem arco	destino arco
1	1	2
2	2	3
3	2	4
4	3	2
5	3	4
6	4	5
7	5	3
8	5	6
9	6	4
10	6	7
11	7	3

Tabela 2.6. Exemplo de lista de arcos ordenada pelas origens.

Dado que todos os arcos com a mesma origem ocupam linhas contíguas, para obter a informação dos arcos com origem num determinado nodo, apenas é necessário conhecer o índice da linha em que está o primeiro arco que nele tem origem e o índice da linha em que começam os arcos com origem no nodo seguinte. Por exemplo, os arcos com origem em 5 começam no índice 7 e os arcos com origem em 6 começam no índice 9. Logo nas posições 7 e 8 estão arcos com origem em 5.

A implementação deste raciocínio resulta em dois vectores: `apontador` e `nodo_destino`. Em `apontador(i)` está a posição do vector `nodo_destino` onde começam os arcos com origem em `i`, como exemplificado na Tabela 2.7. Por exemplo, `apontador(3)=4` significa que o primeiro arco com origem em `3` está na posição `4` de `nodo_destino`. Como `nodo_destino(4)=2`, trata-se do arco (3,2).

	apontador
1	1
2	2
3	4
4	6
5	7
6	9
7	11
8	12

	nodo_destino	Parâmetro
1	2	10
2	3	9
3	4	12
4	2	13
5	4	16
6	5	9
7	3	15
8	6	12
9	4	21
10	7	14
11	3	9

Tabela 2.7. Exemplo de representação de uma rede através de uma estrela de sucessores.

Resumindo, todos os arcos das posições `apontador(i)` até `apontador(i+1)-1` têm origem no nodo `i`. A posição 8 em `apontador` serve para estabelecer onde terminam os arcos com origem no último nodo.

Esta representação é muito eficiente em termos de espaço já que ocupa apenas $n+m+1$ elementos (todos diferentes de zero). É ainda eficiente na maior parte das operações, por exemplo a percorrer os arcos que saem de um nodo (embora não seja a percorrer arcos que entram num dado nodo – tal implica uma extensão da representação que não é abordada aqui).

2.4 Exemplos

2.4.1 Redes de telecomunicações

Uma rede de telecomunicações permite a troca de informação entre diferentes terminais (os emissores e receptores dos sinais transmitidos através da rede) através de ligações electromagnéticas ou ópticas estabelecidas entre i) terminais e equipamento de encaminhamento e ii) entre equipamento de encaminhamento.

A rede telefónica e as redes de computadores, em particular a *internet*, são exemplos de redes de telecomunicações.

Rede telefónica

Numa rede telefónica, os terminais correspondem aos telefones e os nodos às centrais que encaminham os telefonemas por ligações por cabo, fibra óptica ou ondas electromagnéticas.

A diferença entre um telefonema de um telefone fixo e um de um telefone celular (telemóvel) reside em, no caso do telemóvel, a ligação ser feita por ondas electromagnéticas para uma estação de base que está ligada a uma central, enquanto no caso do telefone fixo essa

ligação para uma central ser feita por cabo. A partir da central, todos os telefonemas (fixo ou celulares) são encaminhados pela rede através de cabo ou fibra óptica.

Como ilustrado na Figura 2.12, a rede telefónica é composta por vários níveis. Esta estrutura hierárquica torna a gestão de um sistema à escala planetária possível e permite a eficiente comunicação entre todos os terminais (a nível planetário). Por exemplo, um telefonema entre dois terminais ligados à mesma central local, apenas envolve recursos locais: as ligações entre os terminais e a central local. Já um telefonema entre dois terminais ligados a centrais locais diferentes envolve também uma central regional. A hierarquia prossegue até ao nível internacional.

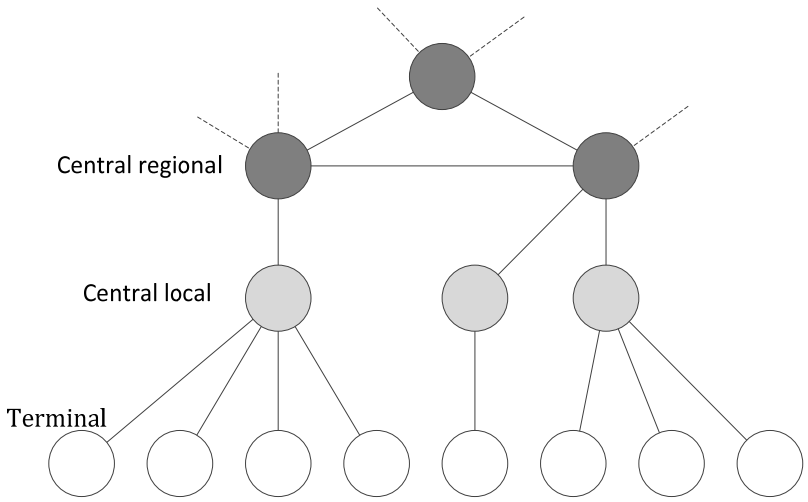


Figura 2.12. Rede telefónica genérica.

Nos primórdios da rede telefónica, um telefonema envolvia o estabelecimento manual de uma ligação, havendo pessoas nas centrais responsáveis por conectar os fios dos dois terminais. Posteriormente, o estabelecimento das ligações passou a ser automático, tendo por base os números de telefone que funcionam como um endereço (por exemplo, 00351 indica Portugal, 21 indica a área de Lisboa, 22 indica a área do Porto, 253 indica a área de Braga e 9 indica uma comunicação móvel).

Por muitos anos, a rede telefónica foi um exemplo de uma rede de comutação de circuitos. Nesse tipo de redes, quando dois terminais estabelecem uma comunicação, é definido um circuito (de facto, seguindo a terminologia de redes, o que é estabelecido é um caminho) por onde passam todos os sinais trocados entre ambos. A comutação de circuitos tem as desvantagens de i) reservar capacidade que não é necessariamente utilizada e ii) bloquear o estabelecimento da comunicação no caso de a rede não ter capacidade suficiente. Por outro lado, a gestão do encaminhamento dos dados e a sua reconstituição nos terminais são simples, o que virtualmente anula o atraso associado à comunicação.

Actualmente, assiste-se à fusão das redes telefónicas com as redes de computadores.

Rede de núcleo e redes de acesso

Uma rede de computadores genérica pode ser dividida na rede de núcleo e em redes de acesso.

A rede de núcleo é responsável pelo encaminhamento do tráfego gerado nas redes de acesso. As ligações entre os nodos da rede de núcleo (nodos de comutação e nodos de comutação de fronteira) são ponto-a-ponto (a transmissão é entre dois e só dois nodos) de alta capacidade (e.g. fibra óptica).

As redes de acesso permitem a ligação de um conjunto de terminais (*hosts* / hospedeiros, *end systems* / sistemas finais, *end users* / utilizadores finais) a nodos de comutação fronteira da rede de núcleo. Uma rede de acesso pode basear-se em ligações partilhadas, ou seja ligações em que o meio de transmissão para o nodo de comutação é usado por mais do que um terminal como acontece, por exemplo, nas redes sem fios. Note-se, no entanto que, em termos da representação numa rede, deve ser considerada uma aresta de cada terminal ao nodo de comutação fronteira (por ser possível a comunicação entre ambos), independentemente da ligação ser partilhada ou ponto-a-ponto.

Na Figura 2.13 representam-se uma rede de núcleo e as redes de acesso a ela ligadas.

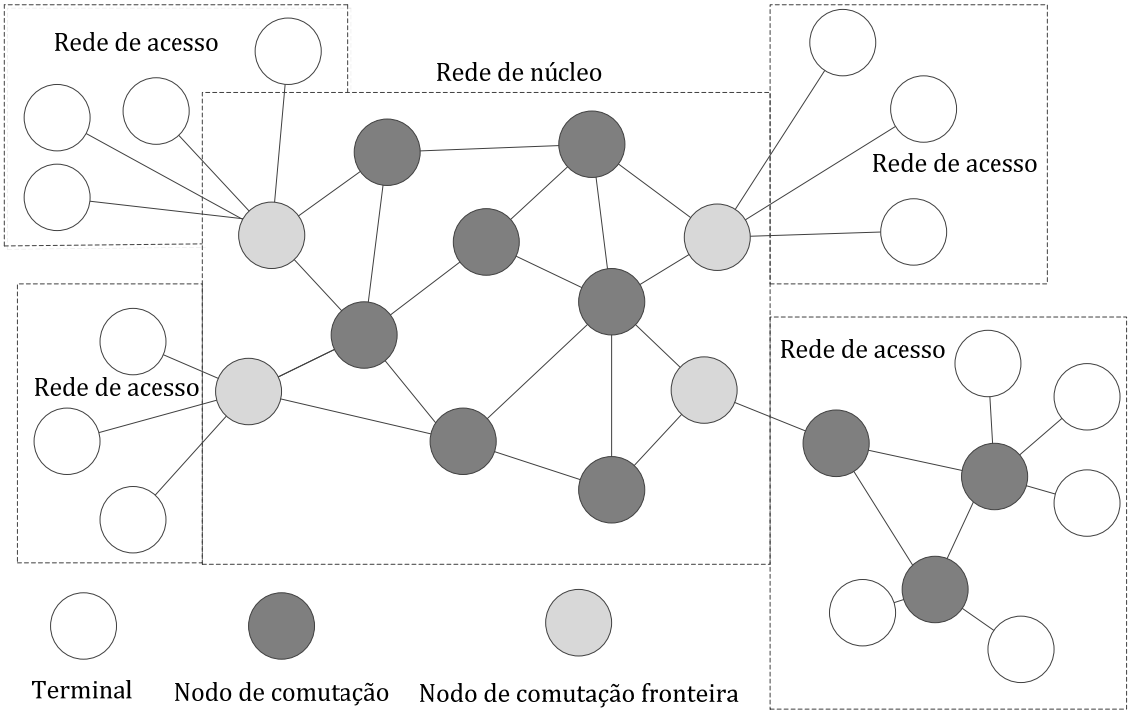


Figura 2.13. Rede de computadores genérica.

Comutação de pacotes

A vasta maioria das modernas redes de computadores usa comutação de pacotes: a comunicação é feita pelo envio de pacotes de dados (sequências de bits com um determinado tamanho máximo que incluem informação sobre a sua origem e destino) de um terminal para outro através de caminhos formados pelos nodos e ligações da rede. Assim, pacotes com a

mesma origem e o mesmo destino podem ser encaminhados por ligações diferentes. Tipicamente, o fluxo de tráfego entre dois terminais é agregado, usando os mesmos caminhos (possivelmente mais do que um) na rede.

Neste mecanismo, o terminal origem junta um cabeçalho (conjunto de bits) à informação a enviar, formando um pacote que é depois encaminhado até ao terminal destino. O cabeçalho contém toda a informação necessária ao encaminhamento (nomeadamente o endereço do terminal destino).

Os terminais são a origem e o destino dos pacotes transmitidos na rede. Os nodos de comutação recebem pacotes e, com base na sua origem e em tabelas de encaminhamento enviam-nos, para outro nodo de comutação ou para o terminal destino. Os nodos de comutação fronteira permitem ligar a rede a outras redes ou terminais.

Fluxo de tráfego e carga

O fluxo de tráfego entre dois terminais é o conjunto de pacotes que o terminal origem envia ao terminal destino ao longo do tempo. A forma mais usual de caracterizar um fluxo de tráfego é o ritmo médio de transmissão medido em bps (*bits per second*) ou Mbps (*megabit* por segundo) (1 Mbps = 1 000 000 bps). O ritmo médio de transmissão é também conhecido por largura de banda.

Os tempos de transmissão e de propagação nas ligações e o tempo de processamento nos nodos são usualmente desprezáveis. O tempo na fila de espera para se processar o encaminhamento num nodo pode ser considerável se a largura de banda dos fluxos de tráfego estiver perto da capacidade da ligação. Para medir a relação entre a largura de banda dos fluxos de tráfego e capacidade da ligação define-se *carga* de uma ligação que corresponde à razão entre a largura de banda dos fluxos de tráfego que atravessam a ligação e a sua capacidade.

Representação em rede

Os terminais, os nodos de comutação e todas as ligações podem ser representados por vértices e arestas. As suas diferentes características serão modeladas através de parâmetros. Por exemplo, a capacidade de uma ligação corresponde à largura de banda que consegue suportar e que será diferente para diferentes tipos de ligação.

Topologias

As redes de computadores podem apresentar diferentes topologias (estruturas das arestas) como exemplificado na Figura 2.14. Numa topologia em estrela existe uma aresta entre um vértice e cada um dos restantes; numa topologia em anel as arestas formam um circuito; numa topologia em árvore existem $n-1$ arestas, em que n é o número de vértices, que não formam ciclos (note-se que a topologia em estrela é um caso particular da topologia em árvore); numa topologia completa, existem $n.(n-1)/2$ arestas, uma para cada par de vértices.

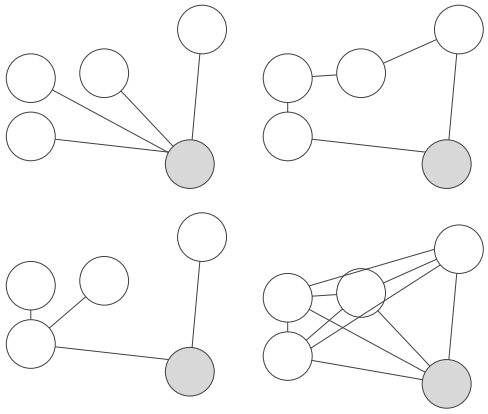


Figura 2.14. Exemplos de topologias de rede, partindo do canto superior esquerdo com a orientação dos ponteiros do relógio: estrela, anel, completa e árvore.

Encaminhamento

Tipicamente, cada nodo de comutação da rede tem uma tabela com, entre outra, a informação de cada arco que existe na rede e o custo a ele associado (definido pelo administrador). Os nodos de comutação comunicam entre si para manterem a tabela com a topologia da rede actualizada. Cada nodo mantém informação actualizada sobre os caminhos mais curtos entre o próprio nodo e todos os outros, ou seja resolvendo um problema de caminho mais curto entre ele próprio e todos os outros. Quando um pacote chega a um nodo i é encaminhado para o primeiro nodo (que não o inicial) do caminho mais curto entre i e o nodo destino do pacote.

Falhas

Em redes de computadores, é usual considerarem-se as implicações de falhas dos arcos, ao contrário das falhas dos nodos paras os quais costumam existir sistemas redundantes. Existem duas formas de protecção de tráfego a falhas nos arcos: baseada em diversificação e baseada em protecção de tráfego.

Na primeira, o tráfego encaminhado por p caminhos disjuntos. Em caso de falha, perde-se d/p de tráfego em que d é a largura de banda do fluxo de tráfego.

Na segunda, o tráfego é encaminhado por $p-1$ caminhos disjuntos, existindo um caminho de protecção que apenas encaminha tráfego em caso de falha. Em caso de falha não se perde tráfego.

Dois casos particulares de protecção de tráfego baseada em caminhos de protecção são o 1:1 (um-para-um) e o 1+1. No método 1:1 existe um caminho por onde é enviado o tráfego e outro caminho alternativo. No método 1+1 o tráfego é duplicado e encaminhado por dois caminhos disjuntos.

Problemas de optimização relevantes

Em redes de telecomunicações, há dois tipos de problemas de optimização que são relevantes: os problemas de desenho/dimensionamento da rede e os problemas de engenharia de tráfego. Nos primeiros pretende-se definir os recursos da rede (nodos e ligações) de forma a minimizar custos (que são usualmente divididos em custos de aquisição da rede (CAPEX) e em

custos de operação e manutenção da rede (OPEX)), suportando o tráfego estimado e satisfazendo requisitos como, por exemplo, os relativos a sobrevivência a falhas.

Nos segundos, assume-se que os recursos da rede estão instalados, pretendendo-se definir o encaminhamento dos diversos fluxos de tráfego. Exemplos de objectivos possíveis são a maximização da utilização da rede e a maximização da robustez da rede a variações imprevistas de tráfego. O encaminhamento pode ser com ou sem bifurcação. Ao contrário do encaminhamento com bifurcação, no encaminhamento sem bifurcação, o fluxo de tráfego entre uma origem e um destino usa apenas um caminho.

2.4.2 Redes sociais

Redes sociais são redes em que um vértice corresponde a uma pessoa, ou grupo de pessoas, ou organização, e uma aresta corresponde a uma interacção entre as entidades representadas pelos vértices.

A título exemplificativo, referem-se três conceitos relevantes no estudo de redes sociais: centralidade, efeito de pequeno mundo e cluster.

Centralidade

A centralidade de um vértice corresponde à sua importância na rede.

Dois exemplos de medidas de centralidade são o grau e a centralidade intermédia. O grau de um vértice é o número de arestas que nele incidem. Um vértice com um grau muito elevado, quando comparado com os restantes, designa-se por hub.

A centralidade intermédia (*betweenness centrality*) de um vértice é definida considerando todos os caminhos mais curtos entre todos os pares de vértices. A centralidade intermédia do vértice é a razão entre o número desses caminhos mais curtos que incluem o vértice e o número total de caminhos mais curtos. Mais formalmente, a centralidade intermédia de um vértice i é dada por

$$c_i = \sum_{st} \frac{n_{st}^i}{g_{st}}$$

onde n_{st}^i é o número de caminhos mais curtos entre os vértices s e t que incluem o vértice i e g_{st}é o número de caminhos mais curtos entre os vértices s e t.

Um exemplo frequentemente usado da utilização de medidas de centralidade é o das relações de influência entre as principais famílias florentinas do início do século XV. A rede correspondente é apresentada na Figura 2.15. O maior grau e a maior centralidade são do vértice 10 que é o associado à família efectivamente mais poderosa: os Medici.

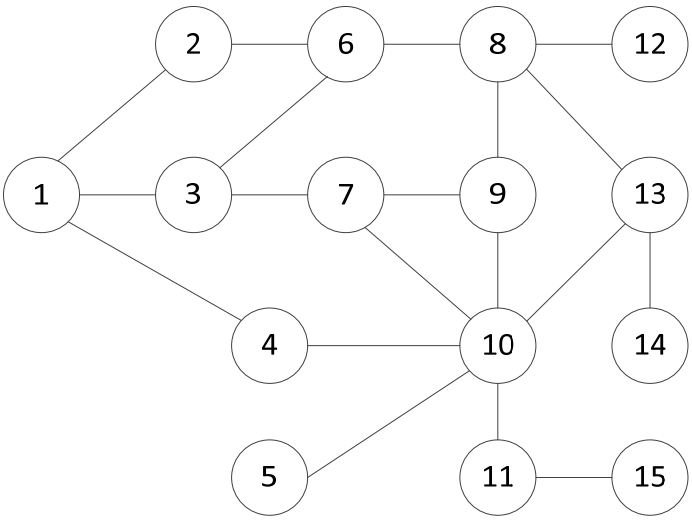


Figura 2.15. Exemplo de rede social.

Efeito de pequeno mundo

O segundo conceito é o efeito pequeno mundo, que ocorre em muitas redes, e que se traduz na distância entre pares de vértices ser reduzida e aumentar pouco com o aumento do número de vértices da rede. A medida usada para a distância é o número de arestas do caminho mais curto entre os dois vértices. Um exemplo de uma rede em que o efeito pequeno mundo se verifica é a internet, onde, tipicamente, um pacote dá apenas 10 a 20 saltos até chegar ao destino.

Cluster

O terceiro conceito é o cluster e corresponde a agrupamentos de vértices com fortes ligações entre si.

Nos exemplos referidos, a optimização desempenha-se um papel importante na determinação de caminhos mais curtos e na definição de clusters.

2.4.3 Transplantes renais cruzados

Em anos recentes têm sido desenvolvidos em vários países programas que têm como objectivo permitir o transplante renal a pessoas que têm alguém (normalmente um familiar próximo) que pretende doar um rim mas com quem não são compatíveis. Considere-se um desses pares em que o dador D1 pretende doar um a rim a um receptor R1 mas que o transplante não é possível, por exemplo por causa da incompatibilidade dos seus tipos sanguíneos. Se existir outro par, D2-R2, e D2 for compatível for R1 e D1 for compatível com R2, então é possível, trocando os dadores, efectuar dois transplantes e R1 e R2 receberem rins. Esta situação é ilustrada na Figura 2.16.

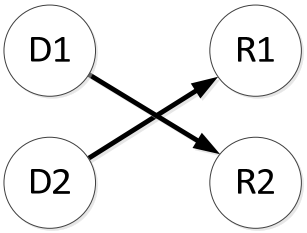


Figura 2.16. Transplantes cruzados.

Este esquema pode ser estendido para mais do que dois transplantes, como ilustrado na Figura 2.17. Nessa figura, cada arco representa a compatibilidade entre um dador e um receptor. Os arcos carregados representam transplantes a efectuar: o dador do par 1 fornece um rim ao receptor do par 3, o dador do par 3 fornece um rim ao receptor do par 2, o dador do par 2 fornece um rim ao receptor do par 1. São assim feitos três transplantes que garantem que em todos os pares envolvidos o dador dá um rim e o receptor recebe um rim (mas não do seu dador original com o qual é incompatível).

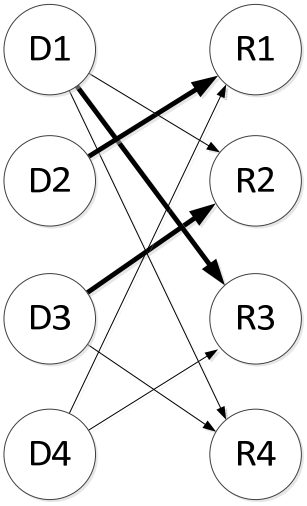


Figura 2.17. Problema com quatro pares de dadores representando numa rede em que cada nodo está associado a um dador ou receptor.

Na Figura 2.18, representa-se o mesmo problema e a mesma solução, mas de forma diferente. Cada nodo está associado a um par incompatível e cada arco *ij* representa que o dador do par *i* é compatível com o receptor do par *j*. Os arcos a carregado representam o mesmo conjunto de transplantes que os arcos carregados da Figura 2.17. Estes arcos formam um circuito.

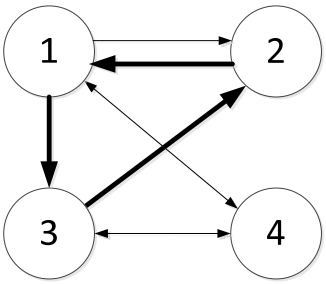


Figura 2.18. Problema com quatro pares de dadores representando numa rede em que cada nodo está associado a par dador-receptor.

Na Figura 2.19 apresenta-se uma instância do problema com 9 pares. Um conjunto de transplantes renais possíveis é um conjunto de circuitos que não têm nodos em comum, tal como exemplificado com os dois conjuntos de transplantes cruzados representadas na Figura 2.20.

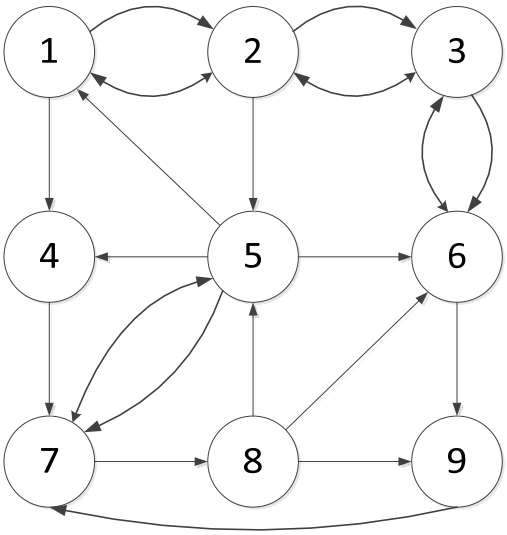


Figura 2.19. Instância do problema dos transplantes renais cruzados com 9 pares.

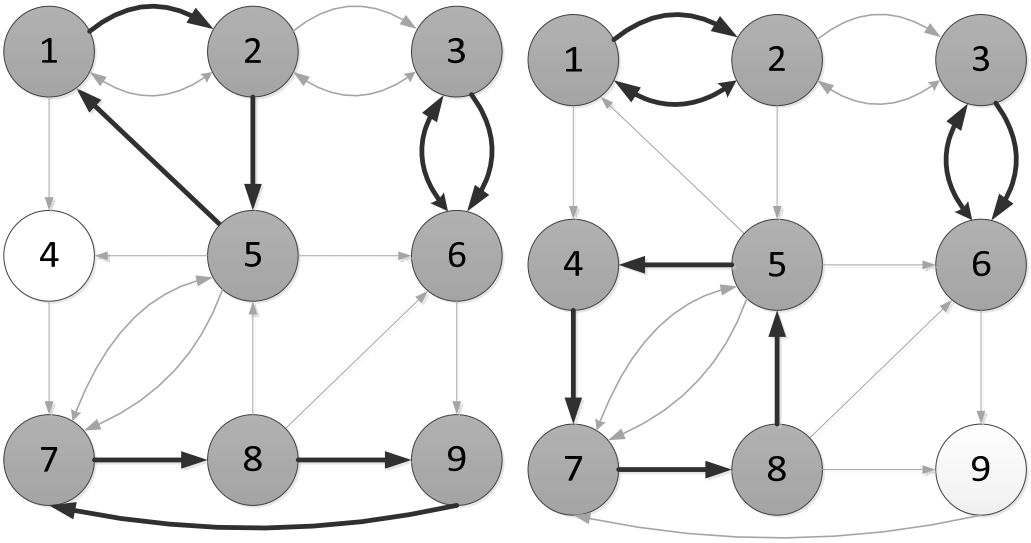


Figura 2.20. Dois conjuntos de transplantes cruzados possíveis.

A optimização tem um papel central nos programas de transplantes renais cruzados ao permitir obter conjuntos de transplantes renais cruzados que maximizam o número de transplantes efectuados, ou outros objectivos relevantes. De acordo com a representação em rede apresentada o problema de maximizar o número de transplantes é equivalente ao problema de seleccionar um conjunto de circuitos (um circuito é conjunto de arcos sucessivos

com início e final no mesmo nodo) disjuntos nos nodos (sem nodos em comum) cujo número de nodos é o maior possível.

2.4.4 Escalonamento de projectos

A abstracção do conceito de rede permite tratar problemas em que não existe um transporte físico de entidades. Por exemplo, em gestão de projectos é frequente utilizar modelos de rede para representar relações temporais entre as diversas actividades que constituem o projecto.

O problema de escalonamento de projectos base consiste em determinar os momentos em que cada actividade de um projecto deve ser iniciada de forma ao projecto ter a menor duração possível, respeitando as relações de precedência entre as actividades.

A definição de um problema de escalonamento de projectos implica a decomposição do projecto conjunto de actividades, a atribuição de uma duração a cada actividade, e o estabelecimento de relações de precedência entre (algumas) actividades. A relação de precedência mais comum é a que força a que uma actividade só se possa iniciar depois de outra estar concluída.

O problema de escalonamento de projectos pode ser modelado numa rede da forma que agora se descreve.

São definidas duas actividades adicionais que correspondem ao início (actividade que precede todas as outras) e ao final (actividade que sucede a todas as outras) do projecto. A cada actividade é associado um nodo. A cada relação de precedência é associado um arco orientado (a actividade origem precede a actividade destino) com um valor correspondente à duração da actividade da origem. São criados arcos com duração 0 entre o nodo da actividade início e cada nodo de cada actividade sem precedentes. São criados arcos entre cada nodo de cada actividade sem sucessores e o nodo da actividade final com a duração da actividade sem sucessores.

O problema consiste em determinar qual a menor duração possível do projecto e pode ser resolvido como um problema de caminho mais longo numa rede acíclica (se houver ciclos as relações de precedência das actividades do projecto estão mal definidas − uma actividade não pode simultaneamente preceder e suceder a outra actividade). Um exemplo de uma rede de escalonamento de projectos é dada na Figura 2.21.

A justificação para se tratar do caminho mais longo é que dado que se pretende acabar o projecto o mais cedo possível, só faz sentido que uma actividade comece a ser executada logo que todas as suas precedentes estejam concluídas. Assim, a data de início de cada actividade i corresponde à distância (duração) do caminho mais longo entre o nodo inicial e o nodo i.

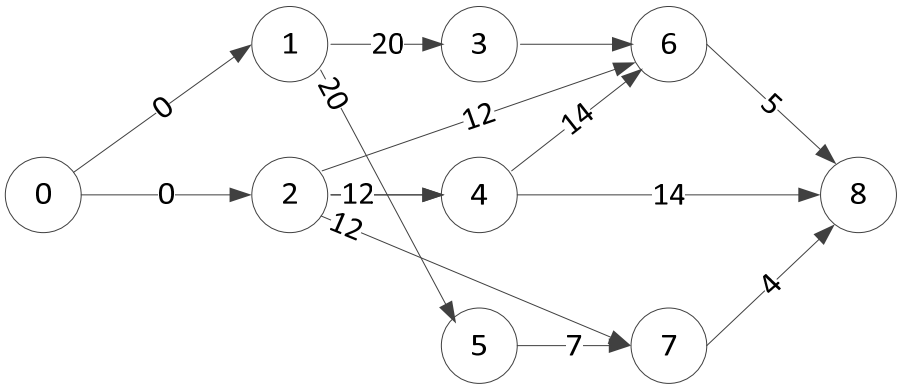


Figura 2.21. Exemplo de uma rede de escalonamento de projectos.

2.4.5 Cadeia de abastecimento

Uma cadeia de abastecimento é um sistema de organizações, pessoas, actividades, informação e recursos, envolvidos no fornecimento de bens ou produtos para satisfação de uma procura.

Na Figura 2.22, é dado um exemplo de uma cadeia de abastecimento com quatro níveis (fornecedores, fábricas, centros de distribuição e clientes), notando-se que uma cadeia de abastecimento pode ter uma estrutura mais complexa, nomeadamente se se tiver em conta a rede inversa que modela a reciclagem dos produtos.

A optimização desempenha um papel relevante em dois tipos de decisões: no desenho da rede e na definição dos fluxos na rede. Os problemas do primeiro tipo incluem a escolha de fornecedores e a definição da localização de fábricas e centros de distribuição, bem como o seu dimensionamento. Os problemas do segundo tipo incluem a definição das quantidades a transportar / distribuir entre as diferentes entidades da cadeia de abastecimento e os modos utilizados. Os problemas de optimização da cadeia de abastecimento têm tipicamente o objectivo da minimização do custo total, tendo em conta o impacto ambiental e satisfazendo níveis de serviço.

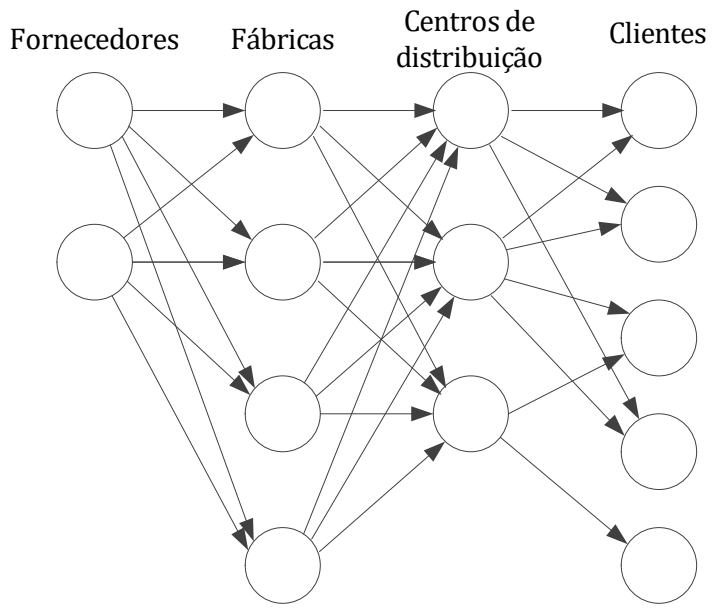
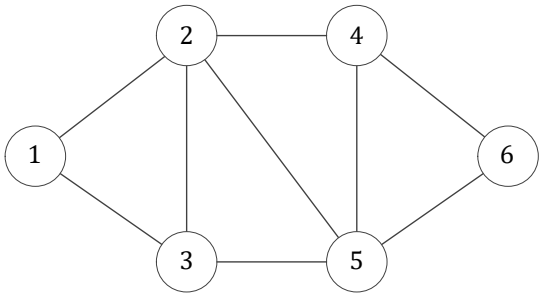


Figura 2.22. Exemplo de uma rede que representa uma cadeia de abastecimento.

2.5 Exercícios

Exercício 2.1 Definições - rede não orientada

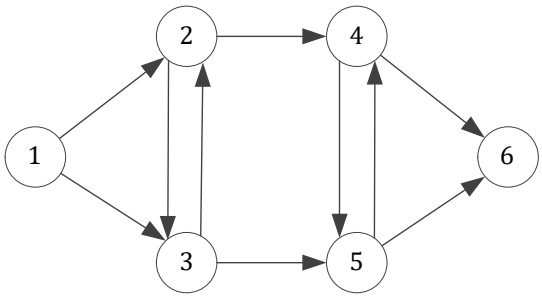
Considere o grafo da figura.



Indique:
a) O grau do vértice 4;
b) Um vértice com grau 2;
c) Um circuito Hamiltoniano;
d) Um circuito Euleriano ou uma prova de que não existe;
e) Uma árvore que não seja de suporte;
f) Uma árvore de suporte.

Exercício 2.2 Definições - rede orientada

Considere o grafo orientado da figura.



Indique:

a) Um caminho não orientado e um caminho orientado.

b) Um circuito que não seja orientado e um circuito orientado.

c) Represente o grafo através de:

i. matriz de adjacência;

ii. matriz de incidência;

iii. estrela de sucessores.

Exercício 2.3 Eficiência espacial

A representação mais eficiente em termos de espaço é

a) estrela de sucessores;
b) matriz de adjacências;
c) matriz de incidência;
d) são todas equivalentes.

Exercício 2.4 Representações em computador

Escrever, em pseudo-código, um algoritmo que tenha como entrada uma rede com custos nos arcos e um determinado nodo e como saída a soma dos custos dos arcos que saem do nodo dado, estando a rede representada através de uma

a) matriz de adjacência;
b) matriz de incidência;
c) estrela de sucessores.

2.6 Bibliografia

• Ahuja, Ravindra K., Thomas L. Magnanti, and James B. Orlin. Network flows. Prentice Hall, 1993.

• Gouveia, Luís, Pedro Moura, Pedro Patrício, Amaro de Sousa. Problemas de Otimização em Redes de Telecomunicações, Faculdade de Ciências da Universidade de Lisboa, 2011.

• Newman, Mark. Networks: an introduction. OUP Oxford, 2010.

• Van Steen, Maarten. Graph theory and complex networks: an introduction, 2010. [Disponível online].

2.7 Resultados de aprendizagem

1. Identificar áreas de aplicação de conceitos e ferramentas relacionados com redes.
2. Descrever os principais termos de redes.
3. Representar redes através de estruturas de dados.

3
Caminhos

3.1 Introdução

O problema de determinar o caminho mais curto entre dois nodos de uma rede, e as suas variantes e extensões, desempenha um papel nuclear em optimização de redes.

O conceito de caminho mais curto permite modelar o percurso de menor distância entre dois pontos mas também a forma mais rápida de chegar a um ponto partindo de outro, ou a mais económica, ou a que usa menos ligações, ou a mais fiável. A rede sobre a qual é definido o problema de caminho mais curto não tem necessariamente correspondência com uma rede física, mas pode ser conceptual (por exemplo, uma rede social, a rede das actividades de um projecto e as suas relações de precedência ou uma rede de programação dinâmica).

Três exemplos de aplicações concretas que envolvem a resolução de problemas de caminho mais curto são: o encaminhamento de um fluxo de tráfego entre dois terminais de uma rede de computadores, o encaminhamento de um veículo entre duas entidades de uma cadeia de abastecimento (e.g. fábrica e armazém), a determinação da menor duração de um projecto e as suas actividades críticas.

Em termos teóricos, os problemas de caminho mais curto são problemas fundamentais em optimização combinatória e ciências de computação. Em termos de aplicações, entre muitas outras áreas, referem-se a logística e as redes de computadores.

Os primeiros algoritmos para problemas de caminho mais curto são dos anos 1950 (Shimbel 1954, Bellman 1956, Ford 1956, Leyzorek et al. 1957, Dijkstra 1959, Moore 1959). Para uma análise mais detalhada dos algoritmos específicos aqui discutidos sugere-se o livro de Ahuja, Magnanti e Orlin (1993).

Neste capítulo aborda-se o problema do caminho mais curto entre dois nodos, bem como variantes e extensões. Para diversos problemas são apresentados modelos de programação inteira e algoritmos específicos.

3.2 Caminho mais curto

3.2.1 Definições e notação

Numa rede orientada, um caminho (elementar ou simples) é uma sequência de nodos distintos e dos arcos que os unem, em que o nodo destino de um arco é nodo origem do arco

seguinte (excepto para o último arco). Usualmente, para simplificação da notação, representam-se os caminhos apenas como uma sequência de nodos. Esta representação não implica perda de informação sobre o caminho se não se permitirem anéis nem arcos múltiplos como acontece neste texto.

O problema do caminho mais curto consiste em, dada uma rede orientada em que a cada arco está associado um custo (ou distância) e dados um nodo inicial e um nodo final, determinar um caminho cuja soma dos custos dos seus arcos tem um valor igual ou inferior à soma dos custos dos arcos de qualquer outro caminho entre os dois nodos em causa. No caso de se ter uma rede não orientada, deve primeiro transformar-se a rede não orientada em orientada substituindo cada aresta por dois arcos com orientações opostas, cada um deles com o mesmo custo do que a aresta inicial.

Formalmente, os parâmetros que definem um problema do caminho mais curto entre dois nodos são os seguintes:

- $G = (N, A)$ grafo orientado
- c_{ij} custo unitário do arco de i para j, $\forall ij \in A$
- o nodo inicial (origem ou fonte)
- d nodo final (destino ou poço)

A rede da Figura 3.1 e a definição do nodo inicial como sendo o nodo 1 e do nodo final como sendo o nodo 10, definem uma instância do problema do caminho mais curto entre dois nodos. Uma solução admissível para esta instância é o caminho $1 - 2 - 6 - 9 - 10$ com custo 13.

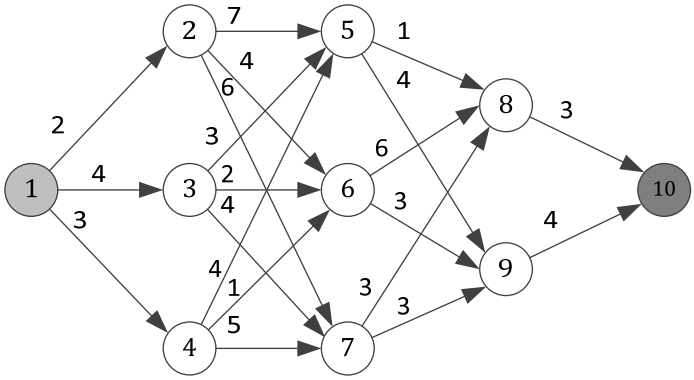


Figura 3.1. Instância do problema do caminho mais curto entre dois nodos (a cinzento claro o nodo inicial e a cinzento escuro o nodo final).

3.2.2 Programação inteira

Uma forma de obter um caminho mais curto entre dois nodos é construir um modelo de programação inteira e, de seguida, aplicar algoritmos gerais para programação inteira que permitem a obtenção de uma solução óptima. Estes algoritmos para programação inteira baseiam-se no método de partição e avaliação ("branch-and-bound") e estão implementados em variadíssimos softwares: por exemplo, *solver* e *openSolver* para *Excel* e *IBM ILOG CPLEX Optimization Studio*.

A existência de algoritmos específicos para o problema mais eficientes do que os algoritmos genéricos para programação inteira não invalida a utilidade da modelação do(s) problema(s) através de programação inteira. A principal razão é o facto de um modelo de programação inteira poder incorporar múltiplas variantes e extensões ao contrário de um algoritmo específico que, por definição, deixa de ser estar correcto quando aplicado a variantes ou extensões do problema.

É agora apresentado o modelo de programação inteira para a instância apresentada acima.

Variáveis de decisão:

$$x_{ij} = \begin{cases} 1 \; se \; arco \; ij \; faz \; parte \; do \; caminho \; mais \; curto \\ 0 \; caso \; contrário \end{cases}, \forall ij \in A$$

Modelo de programação inteira:

$$Min\; z = 2x_{12} + 4x_{13} + 3x_{14} + 7x_{25} + \cdots + 4x_{9,10}$$

$$sujeito\; a:$$

$$x_{12} + x_{13} + x_{14} = 1$$

$$-x_{12} + x_{25} + x_{26} + x_{27} = 0$$

$$-x_{13} + x_{35} + x_{36} + x_{37} = 0$$

$$-x_{14} + x_{45} + x_{46} + x_{47} = 0$$

$$-x_{25} - x_{35} - x_{45} + x_{58} + x_{59} = 0$$

$$-x_{26} - x_{36} - x_{46} + x_{68} + x_{69} = 0$$

$$-x_{27} - x_{37} - x_{47} + x_{78} + x_{79} = 0$$

$$-x_{58} - x_{68} - x_{78} + x_{8,10} = 0$$

$$-x_{59} - x_{69} - x_{79} + x_{9,10} = 0$$

$$-x_{8,10} - x_{9,10} = -1$$

$$x_{ij} \in \{0,1\}, \forall ij \in A$$

A função objectivo traduz o custo de um caminho em função dos arcos que o compõem. O modelo tem uma restrição para cada nodo. Considere-se um nodo que não o inicial nem o final, a sua restrição força a que se o caminho inclui um arco que entra no nodo então também tem de incluir um arco que sai do nodo (e vice-versa). Se o caminho não inclui nenhum arco que entra do nodo então também não pode inclui nenhum arco que sai (e vice-versa).

A restrição do nodo inicial (a primeira) força a que o caminho inclua um arco que saia desse nodo e a restrição do nodo final (a última antes das restrições de domínio das variáveis de decisão) força a que o caminho inclua um arco que chegue a esse nodo.

As restrições agora descritas designam-se por restrições de conservação de fluxo.

As restrições de domínio das variáveis de decisão implicam que um arco ou faz parte do caminho ou não ou, de outra forma, um caminho não pode incluir fracções de arcos.

É agora apresentado um modelo de programação inteira para o problema geral do caminho mais curto entre dois nodos.

Variáveis de decisão:

$$x_{ij} = \begin{cases} 1 \; se \; arco \; ij \; faz \; parte \; do \; caminho \; mais \; curto \\ 0 \; caso \; contrário \end{cases}, \forall ij \in A$$

Modelo de programação inteira:

$$Min\ z = \sum_{ij \in A} c_{ij} x_{ij}$$

$$sujeito\ a:$$

$$\sum_{j:ij \in A} x_{ij} - \sum_{j:ji \in A} x_{ji} = \begin{cases} 1, se\ i = o \\ 0, se\ i \neq s, i \neq t \\ -1, se\ i = d \end{cases}, \forall i \in N$$

$$x_{ij} \in \{0,1\}, \forall ij \in A$$

É importante notar que as restrições de domínio podem ser substituídas por

$$0 \leq x_{ij} \leq 1, \forall ij \in A$$

O modelo resultante desta substituição designa-se por relaxação linear. Para obter a relaxação linear de um modelo, removem-se as restrições de integralidade das variáveis. Demonstra-se que, nalguns modelos, como é o caso do modelo apresentado para o problema do caminho mais curto, existe sempre uma solução óptima da relaxação linear em que as variáveis tomam valores inteiros (neste caso, binários, $x_{ij} = 0$ ou $x_{ij} = 1$) pelo que a resolução da relaxação linear equivale à resolução do problema inteiro. A importância desta observação é que os modelos de programação linear (como a relaxação linear referida) são, virtualmente sempre, muito mais fáceis de resolver do que modelos de programação inteira cuja solução óptima da relaxação linear não é necessariamente inteira. De um modelo de programação inteira cuja relaxação linear tem necessariamente uma solução óptima inteira diz-se que tem a propriedade da integralidade.

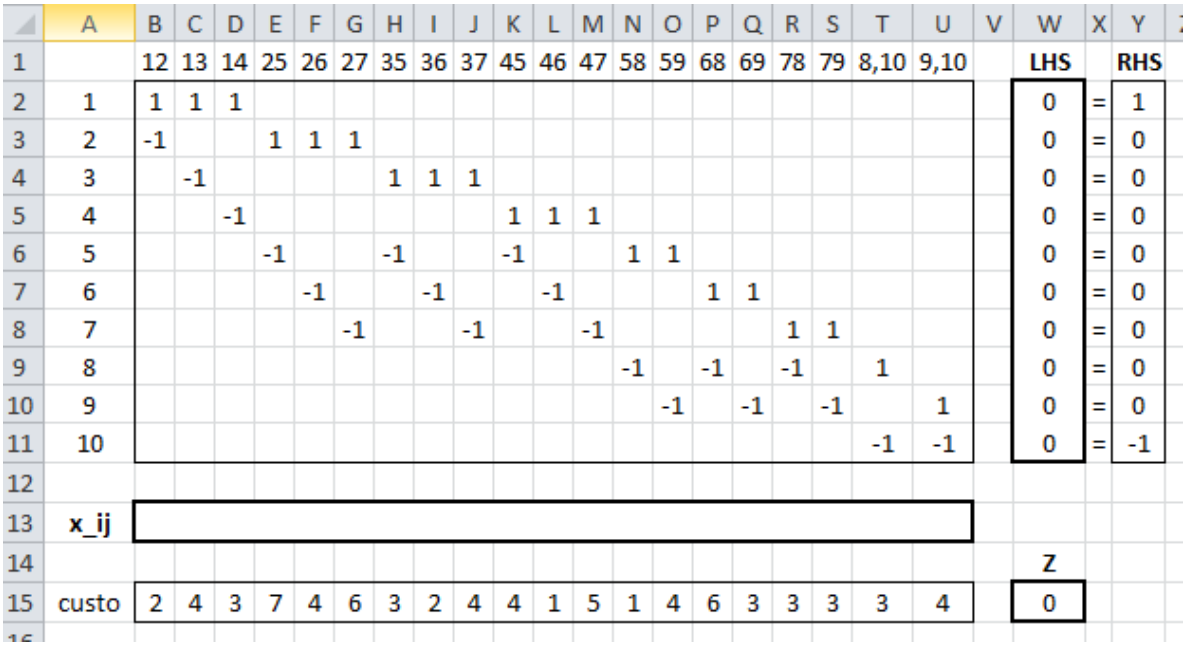
Assumindo que não há ciclos negativos (ciclos cuja soma dos custos é negativa) na rede (ver subsecção 3.2.6, página 52), as restrições de domínio podem ser simplificadas ainda mais já que por causa das restrições de conservação de fluxo, nenhuma variável toma um valor maior que 1 o que torna $x_{ij} \leq 1, \forall ij \in A$ redundante, ficando o domínio das variáveis definido por

$$x_{ij} \geq 0, \forall ij \in A$$

3.2.3 Solver/OpenSolver para Excel

Representando a rede através de uma matriz de incidência (ver página 28), a introdução do modelo no Excel (ou folha de cálculo análoga) é directa. Na Figura 3.2 apresenta-se a introdução do modelo:

- As células da linha x_ij (linha 13) correspondem às variáveis de decisão sendo deixadas em branco (é o *solver* que vai escrever nestas células o valor óptimo destas variáveis).
- As células da coluna LHS (left hand side) correspondem aos membros esquerdos das restrições. Por exemplo, para a primeira célula (célula W2) a fórmula a introduzir é "=SUMPRODUCT(B2:U2;$B$13:$U$13)" e para a última célula (célula W11) a fórmula é "=SUMPRODUCT(B11:U11;$B$13:$U$13)".
- As células da coluna RHS (right hand side) correspondem aos membros direitos das restrições e são constantes.
- A célula Z corresponde à função objectivo, contendo a fórmula "=SUMPRODUCT(B13:U13;B15:U15)".



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1		12	13	14	25	26	27	35	36	37	45	46	47	58	59	68	69	78	79	8,10	9,10		LHS		RHS
2	1	1	1	1																			0	=	1
3	2	-1			1	1	1																0	=	0
4	3		-1					1	1	1													0	=	0
5	4			-1							1	1	1										0	=	0
6	5				-1			-1			-1			1	1									0	=	0
7	6					-1			-1			-1				1	1							0	=	0
8	7						-1			-1			-1					1	1					0	=	0
9	8													-1		-1		-1		1				0	=	0
10	9														-1		-1		-1		1			0	=	0
11	10																			-1	-1			0	=	-1
12																									
13	x_ij																								
14																							Z		
15	custo	2	4	3	7	4	6	3	2	4	4	1	5	1	4	6	3	3	3	3	4		0		

Figura 3.2. Resolução do problema do caminho mais curto com o *Solver*/*OpenSolver* para *Excel*.

Na Figura 3.3 é apresentada a caixa de diálogo de configuração do *Solver* do *Excel*. É de salientar a importância de seleccionar o Simplex LP como método de resolução já que é o modelo a ser resolvido é de programação linear. Este método também deveria ser seleccionado no caso de existirem variáveis inteiras.

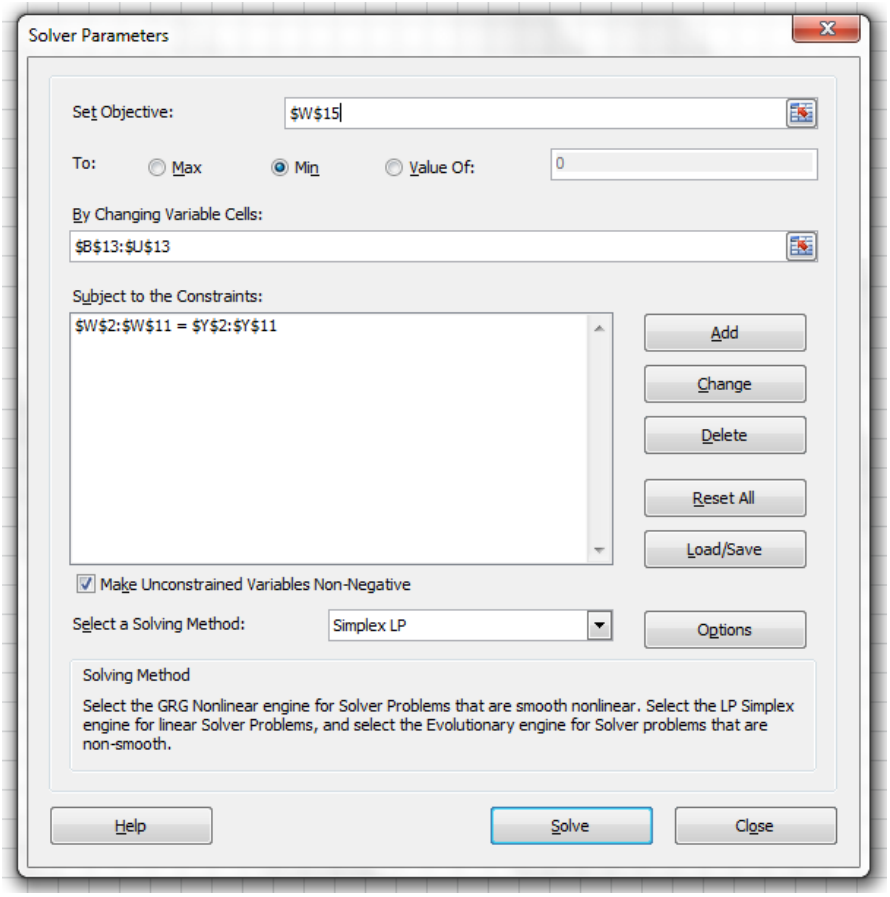


Figura 3.3. Parâmetros do Solver do Excel.

Na Figura 3.4 é apresentada a solução obtida com o *Solver* (sendo importante verificar que após a optimização o *Solver* retornou a indicação de que a solução é óptima).

Assim, uma solução óptima é $x_{13} = x_{35} = x_{58} = x_{8,10} = 1$ e todas as restantes variáveis com valor 0. O valor desta solução é 11. Esta solução não é necessariamente a única solução óptima e, caso exista, o *Solver* pode retornar outra solução óptima de forma arbitrária (por exemplo noutro computador ou com outras versão do *Solver*).

A utilização do *OpenSolver* é muito semelhante à do *Solver*, pelo que não é aqui explorada. Refere-se apenas que para modelos de maior dimensão, o *OpenSolver* é claramente preferível ao *Solver*.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1		12	13	14	25	26	27	35	36	37	45	46	47	58	59	68	69	78	79	8,10	9,10		LHS		RHS
2	1	1	1	1																			1	=	1
3	2	-1			1	1	1																0	=	0
4	3		-1					1	1	1													0	=	0
5	4			-1							1	1	1										0	=	0
6	5				-1			-1			-1			1	1									0	=	0
7	6					-1			-1			-1				1	1							0	=	0
8	7						-1			-1			-1					1	1					0	=	0
9	8													-1		-1		-1		1				0	=	0
10	9														-1		-1		-1		1			0	=	0
11	10																			-1	-1			-1	=	-1
12																									
13	**x_ij**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**				
14																							**Z**		
15	custo	2	4	3	7	4	6	3	2	4	4	1	5	1	4	6	3	3	3	3	4		11		

Figura 3.4. Modelo e solução obtida com o *Solver*.

3.2.4 *IBM ILOG CPLEX Optimization Studio

O conteúdo do ficheiro com o modelo em OPL para o problema do caminho mais curto é o seguinte

```
tuple arc {
  int i;
  int j;
  float cost;
}
int n=...;
int m=...;
int o=...;
int d=...;

arc listArcs[1..m]=...;

dvar float+ x[1..m];

minimize sum (a in 1..m) listArcs[a].cost*x[a];
subject to {
  sum (a in 1..m : listArcs[a].i==o) x[a] == 1;
  sum (a in 1..m : listArcs[a].j==d) -x[a] == -1;
  forall (k in 1..n: k!=o && k!=d)
    sum (a in 1..m : listArcs[a].i==k) x[a] - sum (a in 1..m :
         listArcs[a].j==k) x[a] == 0;
}
```

E o ficheiro de dados deve conter

```
n=10;
m=20;
o=1;
d=10;

listArcs=
[
<    1    ,    2    ,      2      >      ,
<    1    ,    3    ,      4      >      ,
<    1    ,    4    ,      3      >      ,
<    2    ,    5    ,      7      >      ,
<    2    ,    6    ,      4      >      ,
<    2    ,    7    ,      6      >      ,
<    3    ,    5    ,      3      >      ,
<    3    ,    6    ,      2      >      ,
<    3    ,    7    ,      4      >      ,
<    4    ,    5    ,      4      >      ,
<    4    ,    6    ,      1      >      ,
<    4    ,    7    ,      5      >      ,
<    5    ,    8    ,      1      >      ,
<    5    ,    9    ,      4      >      ,
<    6    ,    8    ,      6      >      ,
<    6    ,    9    ,      3      >      ,
<    7    ,    8    ,      3      >      ,
<    7    ,    9    ,      3      >      ,
<    8    ,    10   ,      3      >      ,
<    9    ,    10   ,      4      >
];
```

Caso seja feita a ligação à folha de cálculo do ficheiro "caminhos.xlsx", folha "opl_data" e com os parâmetros da instância nas células com os nomes apresentados, o ficheiro de dados deve conter:

```
SheetConnection sheet("caminhos.xlsx");

n from SheetRead(sheet,"opl_data!n");
m from SheetRead(sheet,"opl_data!m");
o from SheetRead(sheet,"opl_data!o");
d from SheetRead(sheet,"opl_data!d");
listArcs from SheetRead(sheet,"opl_data!listArcs");

x to SheetWrite(sheet,"opl_data!solution");
```

3.2.5 *Caminho mais curto e dualidade

O problema dual do problema de programação linear apresentado é o seguinte.
Variáveis de decisão:

$$d_j - comprimento\ do\ caminho\ mais\ curto$$
$$entre\ o\ nodo\ inicial\ e\ o\ nodo\ j, \forall j \in N \backslash \{0\}$$

Note-se que $d_o = 0$ logo não é considerado uma variável de decisão.
Modelo de programação linear:

$$Max\ w = -d_d$$

$sujeito\ a$:
$d_i - d_j \leq c_{ij}, \forall ij \in A$
$d_j\ livre, \forall j \in N \backslash \{1\}$

3.2.6 *Ciclos negativos

Um ciclo é negativo se a soma dos custos dos arcos que o compõem é negativa. A existência de um ciclo negativo numa rede resulta de existirem lucros associados às actividades relacionadas com os arcos ou com os nodos. Por exemplo, entregar uma encomenda a um cliente pode ser modelado como um lucro associado ao nodo que representa o cliente.

Considere-se a instância do problema de caminho mais curto entre os nodos 1 e 6 apresentada na Figura 3.5.

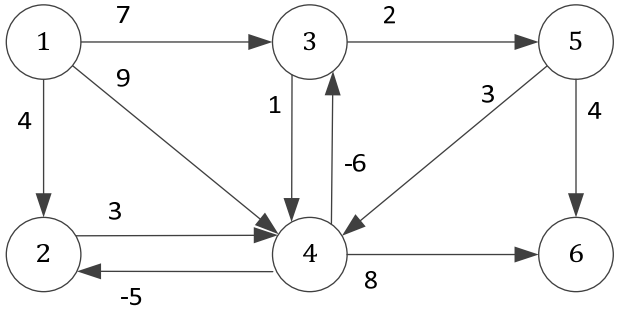


Figura 3.5. Exemplo de uma rede com ciclos negativos.

Se o domínio das variáveis de decisão não for limitado superiormente (i.e., se o domínio for $x_{ij} \geq 0$), o modelo de programação inteira é ilimitado, já que o valor das variáveis de decisão de um ciclo negativo, por exemplo x_{24} e x_{42}, pode aumentar indefinidamente mantendo a validade da solução (as restrições de conservação de fluxo são respeitadas) e diminuindo o valor da função objectivo.

Se o domínio das variáveis de decisão for limitado ($0 \leq x_{ij} \leq 1$), o problema deixa de ser ilimitado mas a solução óptima pode conter ciclos, não se tratando portanto de um caminho elementar (sem repetição de arcos ou nodos) como o desejado (exemplo da Figura 3.6) ou incluindo para além de um caminho um conjunto de ciclos (Figura 3.7).

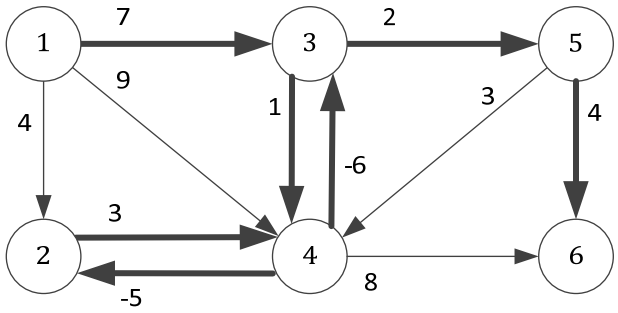


Figura 3.6. Solução óptima de um modelo de programação inteira (arcos do caminho mais curto não elementar a carregado) que não corresponde a um caminho elementar.

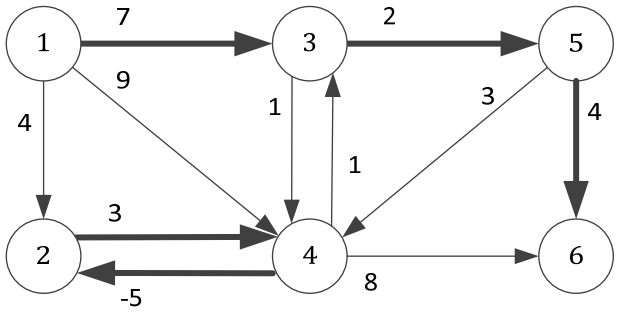


Figura 3.7. Solução óptima de um modelo de programação inteira (arcos do caminho mais curto não elementar a carregado) que não corresponde apenas a um caminho mas a a um caminho e um ciclo.

O problema do caminho mais curto elementar numa rede com ciclos negativos é um problema difícil de modelar e de resolver (é um problema NP-difícil ao contrário do problema de caminho mais curto em redes sem ciclos negativos para o qual existem algoritmos polinomiais). Para o resolver pode ser usada uma estratégia baseada em iterativamente, até ser obtido um caminho sem ciclos, adicionar restrições ao modelo que eliminem os ciclos da solução óptima actual. Por exemplo, a inclusão no modelo da restrição $x_{24} + x_{42} \leq 1$ eliminaria o ciclo $4 - 2 - 4$.

3.2.7 Algoritmo específicos

Na secção seguinte serão apresentados algoritmos para determinar o caminho mais curto entre um nodo e todos os restantes. Esses algoritmos podem ser aplicados no problema de determinar o caminho mais curto entre dois nodos. Nalguns casos (algoritmo para redes acíclicas e Dijkstra), se se pretender apenas o caminho mais curto para um nodo, o algoritmo pode ser interrompido mal este seja encontrado, não sendo necessário determinar todos os caminhos mais curtos.

3.3 Árvore de caminhos mais curtos

O problema de determinar os caminhos mais curtos entre um nodo e todos os restantes é equivalente a $n - 1$ problemas de caminho mais curto entre dois nodos independentes (podem ser resolvidos sequencialmente). No entanto, existem vantagens em termos de modelação e resolução em considerá-lo de forma integrada. Excepto quando referido explicitamente, consideram-se redes sem ciclos negativos[3].

[3] Um ciclo é negativo se a soma dos custos dos arcos que o compõem é negativa.

3.3.1 Conceitos preliminares

Assumindo que existe pelo menos um caminho entre a raiz e cada um dos restantes nodos, o conjunto dos arcos que pertencem aos caminhos mais curtos entre a raiz e os restantes nodos formam uma árvore[4] (ver definição de árvore na página 24). Um exemplo é dado na Figura 3.8 onde os arcos a carregado definem a árvore dos caminhos mais curtos com raiz em 1.

O caminho (único) que liga a raiz da árvore dos caminhos mais curtos e um nodo é o caminho mais curto entre o nodo inicial e esse nodo.

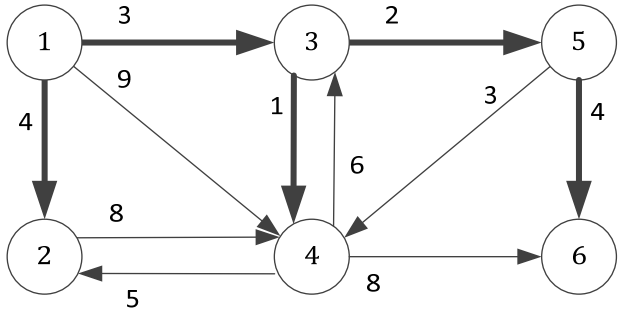


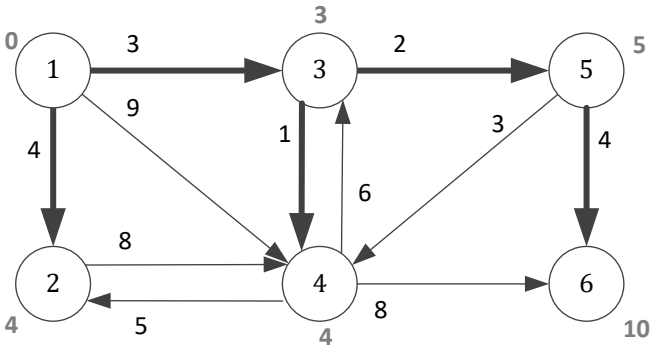
Figura 3.8. Uma árvore de caminhos mais curtos.

Note-se que se os arcos dos caminhos mais curtos não formassem uma árvore, havia necessariamente um ciclo, o que implicaria a existência de mais do que um caminho entre dois nodos, sendo óbvio que apenas o mais curto precisaria de ser considerado.

O comprimento do caminho (único) entre a raiz e cada um dos restantes nodos pode ser determinado através de

$$d_j = d_i + c_{ij}$$

em que d_j é o comprimento do caminho entre a raiz e o nodo $j, \forall j \in N \setminus \{1\}$. Tal cálculo é ilustrado na Figura 3.9 onde os valores de d_j para a árvore representada pelos arcos a carregado são apresentados juntos aos nodos.



[4] De forma mais correcta, mas usualmente não empregue, designa-se esta estrutura por arborescência por o grafo ser orientado e as árvores serem definidas em grafos não orientados.

Figura 3.9. Comprimentos obtidos com base numa árvore.

Note-se que para todos os arcos ij verifica-se que

$$d_j \leq d_i + c_{ij}$$

o que significa que se o comprimento do caminho até i somado com o custo do arco ij não pode ser menor do que o comprimento até j. Se o fosse, o caminho actual até j não seria o mais curto, mais sim o caminho até i com o arco ij.

De forma geral, os valores de d_j correspondem aos comprimentos do caminho mais curto entre o nodo inicial e o nodo $j, \forall j \in N$, se e só se

$$d_j \leq d_i + c_{ij}, \forall ij \in A.$$

Para os arcos que fazem parte do caminho mais curto, verifica-se

$$d_j = d_i + c_{ij}.$$

Dado tratar-se de uma árvore, uma solução para o problema do caminho mais curto entre um nodo e todos os restantes, pode ser representada através de um vector que a cada nodo associa o seu predecessor imediato. No exemplo pred(6)=5, pred(5)=3, pred(4)=3, pred(3)=1 e pred(2)=1.

3.3.2 Programação Inteira

Um modelo de programação inteira para o problema de determinar o caminho mais curto entre um nodo (nodo raiz com índice 1) e todos os restantes, aqui designado por modelo desagregado, baseia-se na seguinte definição de variáveis de decisão:

$$x_{ij}^k = \begin{cases} 1 \; se \; arco \; ij \; faz \; parte \; do \; caminho \; mais \; curto \\ \qquad da \; raiz \; para \; o \; nodo \; k \\ 0 \; caso \; contrário \end{cases}, \forall ij \in A$$

Modelo desagregado:

$$Min \; z = \sum_{k \in N \setminus \{o\}} \sum_{ij \in A} c_{ij} x_{ij}^k$$

$$sujeito \; a:$$

$$\sum_{j:ij \in A} x_{ij}^k - \sum_{j:ji \in A} x_{ij}^k = \begin{cases} 1, \; se \; i = 0 \\ -1, \; se \; i = k \\ 0, \; caso \; contrário \end{cases}, k \in N\{o\}, \forall i \in N$$

$$x_{ij}^k \in \{0,1\}, \forall ij \in A, k \in N \setminus \{o\}$$

As restrições de domínio das variáveis de decisão podem ser substituídas por

$$x_{ij}^k \geq 0, \forall ij \in A, k \in N \setminus \{o\}$$

Um modelo alternativo, com menos variáveis de decisão mas também menos flexibilidade na modelação de variantes e extensões é o modelo agregado.

Variáveis de decisão:

$$x_{ij} - \; número \; de \; caminhos \; que \; incluem \; o \; arco \; ij, \forall ij \in A$$

Modelo agregado:

$$Min \; z = \sum_{ij \in A} c_{ij} x_{ij}$$

$$sujeito \; a:$$

$$\sum_{j:ij\in A} x_{ij} - \sum_{j:ji\in A} x_{ji} = \begin{cases} n-1, \, se \, i = o \\ -1, \, se \, i \neq o \end{cases}, \forall i \in N$$

$$x_{ij} \geq 0 \; e \; inteiros, \forall ij \in A$$

As restrições de domínio das variáveis de decisão podem ser substituídas por

$$x_{ij} \geq 0, \forall ij \in A.$$

3.3.3 Redes acíclicas

Numa rede acíclica os índices dos nodos podem ser tais que o nodo origem de qualquer arco tenha um índice inferior ao seu nodo destino. Formalmente, $i < j, \forall ij \in A$. Se esta propriedade se verificar, diz-se que a rede está ordenada topologicamente. É possível haver mais do que uma ordenação topológica.

Descreve-se agora um algoritmo de ordenação topológica que retorna os índices correspondentes à ordenação topológica obtida ou a indicação de que existe um ciclo (orientado) e que portanto a ordenação topológica não é possível.

```
// Algoritmo de ordenação topológica
// Vector index fica com o novo índice para cada nodo
j=1
Enquanto existirem nodos com grau de entrada = 0
     Seleccionar um nodo i com grau de entrada = 0
     index[i]=j
     Remover da rede todos os arcos com origem em i
     Remover da rede o nodo i
     j=j+1
Se rede está vazia, ordenação topológica concluída
Se não, existe um ciclo (orientado)
```

Ilustra-se agora o algoritmo de ordenação topológica com base na rede da Figura 3.10. A Figura 3.11 mostrando o resultado da aplicação do algoritmo de ordenação topológica que, nesta instância, é uma ordenação incompleta por a rede ter um ciclo e portanto não ser possível a ordenação topológica.

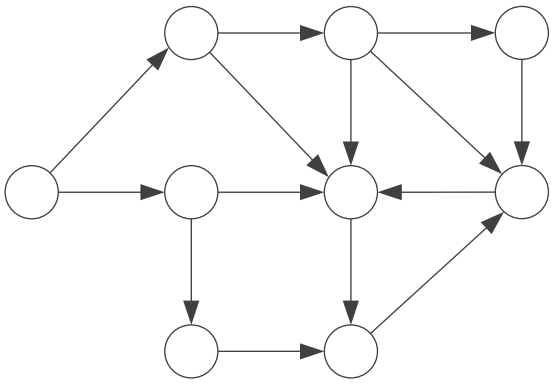


Figura 3.10. Exemplo de instância para ordenação topológica.

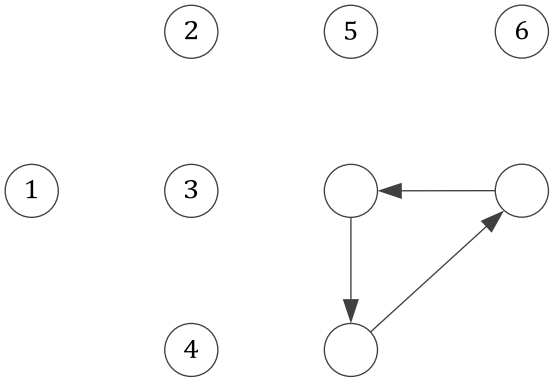


Figura 3.11. Resultado da aplicação do algoritmo de ordenação topológica.

Para obter a árvore dos caminhos mais curtos numa rede ordenada topologicamente, basta percorrer sequencialmente os nodos por onde crescente do índice, e verificar se, para os arcos que saem do nodo da iteração actual, se verifica $d_i + c_{ij} < d_j$. Se não, o comprimento até j deve ser actualizado através de $d_j = d_i + c_{ij}$. Salienta-se que este algoritmo é válido para redes com custos arbitrários.

O algoritmo é apresentado de seguida.

```
// Algoritmo para determinar árvore dos caminhos mais curtos em redes
        acíclicas
// Inicialização
Para todos os nodos j
     dist(j)=INF    // INF constante muito elevada
     pred(j)=0      // não há predecessores
dist(1)=0
// Ciclo
Para i=1 até n
     Para todos os arcos ij
          Se dist(j)>dist(i)+custo(ij)
               dist(j)=dist(i)+custo(ij)
               pred(j)=i
```

Exemplifica-se a aplicação deste algoritmo com a instância da Figura 3.12. A aplicação do algoritmo é mostrada na Tabela 3.1. e a solução obtida na Figura 3.13.

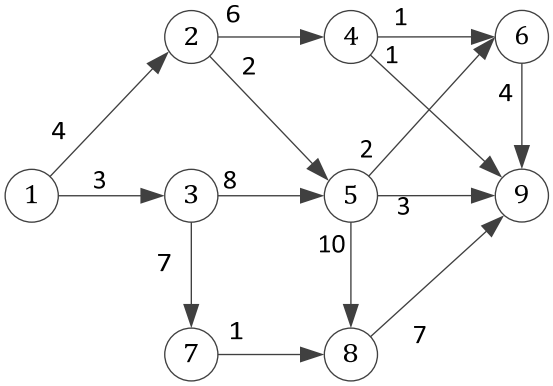


Figura 3.12. Instância de problema do caminho mais curto com rede já ordenada topologicamente.

		Nodo (dist,pred)								
		1	2	3	4	5	6	7	8	9
Iteração	0	(0,0)	(INF,0)	(INF,0)	(INF,0)	(INF,0)	(INF,0)	(INF,0)	(INF,0)	(INF,0)
	1		(4,1)	(3,1)						
	2				(10,2)	(6,2)				
	3					(6,2)		(10,3)		
	4						(11,4)			(11,4)
	5						(8,5)		(16,5)	(9,5)
	6									(9,5)
	7								(11,7)	
	8									(9,5)
	9									

Tabela 3.1. Aplicação do algoritmo numa rede acíclica.

Por exemplo, na iteração 3, para o nodo 5, é testada a condição 6>3+8 que é falsa, logo a distância e o predecessor associados ao nodo 5 mantêm-se. Um outro exemplo, na iteração 5, para o nodo 6, é testada a condição 11>6+2 que é verdadeira, logo há uma actualização da distância e do predecessor do nodo 6.

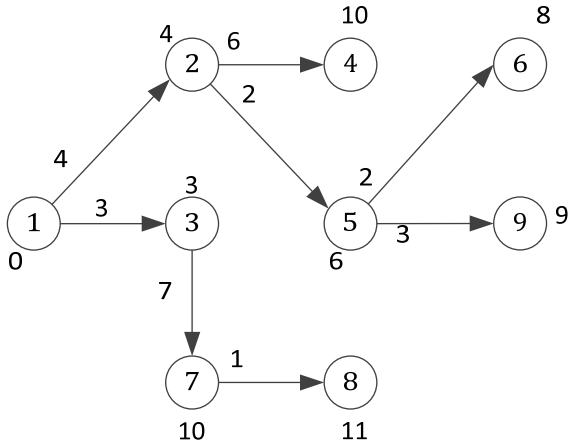


Figura 3.13. Árvore de caminhos mais curtos numa rede acíclica.

3.3.4 Redes sem custos negativos: algoritmo de Dijkstra

O algoritmo de Dijkstra permite a obtenção do caminho mais curto entre um nodo e todos os restantes em redes que podem ter ciclos mas não têm sem custos negativos.

Em cada iteração do algoritmo de Dijkstra há um nodo para o qual se tem a certeza que o caminho mais curto foi encontrado, ficando a sua distância inalterável até à conclusão do algoritmo. Diz-se que esse nodo tem uma etiqueta permanente.

Em cada iteração, o nodo que passa a ter a etiqueta permanente é aquele que tem uma menor distância de entre todos os que têm etiquetas temporárias (por oposição a permanentes).

Na Tabela 3.2 apresenta-se a aplicação do algoritmo de Dijsktra à instância da Figura 3.14.

```
// Algoritmo de Dijkstra para o problema do caminho mais curto entre um
        nodo e todos os restantes em redes sem custos negativos
// S é conjunto dos nodos para os quais o caminho mais curto já foi
        encontrado
// Inicialização
Para todos os nodos j
     dist(j)=INF    // INF constante muito elevada
     pred(j)=0      // não há predecessores
dist(1)=0
S={}
// Ciclo
Enquanto S não contiver todos os nodos
     Seleccionar o nodo i que não pertence a S e que tem menor de dist
     Adicionar i ao conjunto S
     Para todos os arcos ij
          Se dist(j)>dist(i)+custo(ij)
               dist(j)=dist(i)+custo(ij)
               pred(j)=i
```

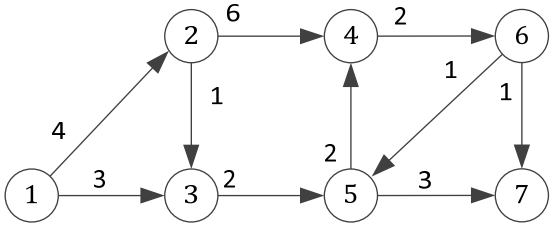


Figura 3.14. Instância para aplicação do algoritmo de Dijkstra.

Iteração	i	Nodo (dist,pred)						
		1	2	3	4	5	6	7
0	-	(0,0)	(INF,0)	(INF,0)	(INF,0)	(INF,0)	(INF,0)	(INF,0)
1	1	-	(4,1)	(3,1)				
2	3	-	(4,1)	-		(5,3)		
3	2	-	-	-	(10,2)	(5,3)		
4	5	-	-	-	(7,5)	-		(8,5)
5	4	-	-	-	-	-	(9,4)	(8,5)
6	7	-	-	-	-	-	(9,4)	-
7	6	-	-	-	-	-		-

Tabela 3.2. Aplicação do algoritmo de Dijkstra. As iterações em que as etiquetas de cada nodo se tornam permanentes têm os limites carregados. Por exemplo, na iteração 3, a etiqueta do nodo 5 passa a permanente porque é do nodo 5 a menor distância (de valor 5) quando comparada com os restantes nodos com etiquetas temporárias: nodos 4 (distância 10), 6 e 7 (distâncias INF).

3.3.5 *Redes com custos arbitrários: algoritmo de Bellman-Ford

No caso de existirem custos negativos, o algoritmo de Dijkstra deixa de ser válido, já que não é possível garantir a obtenção de um caminho mais curto para um nodo em cada iteração.

Um pequeno exemplo desse caso é dado na Figura 3.15. Embora o nodo 3 tenha o menor comprimento na primeira iteração, esse comprimento ainda pode diminuir.

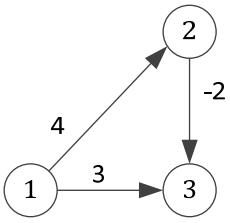


Figura 3.15. Exemplo de rede com custos negativos.

O algoritmo que lida com custos negativos é o algoritmo de Bellman-Ford. Note-se, no entanto, que se houver ciclos negativos, é necessária a aplicação de algoritmos mais elaborados, limitando-se o algoritmo de Bellman-Ford a identificar a existência de um ciclo negativo.

No algoritmo de Bellman-Ford, é mantida uma lista de nodos que podem conduzir à actualização da distância de outros nodos. Um nodo pode entrar e sair dessa lista mais de uma vez. Um nodo entra na lista de cada vez que a sua distância é actualizada e sai da lista de cada vez que é analisado.

Dado poderem existir arcos de custo negativo, a distância de um nodo pode ser negativa. No entanto, um caminho elementar nunca terá uma distância menor que nC em que C é o menor custo de um arco. Dessa forma, se durante a execução do algoritmo uma distância tomar um valor menor que nC, foi detectada a existência de um ciclo negativo (que pode ser identificado, tópico que não é aqui coberto).

```
// Algoritmo de Bellman-Ford para o problema do caminho mais curto entre
        um nodo e todos os restantes em redes com custos arbitrários
// L é conjunto dos nodos a analisar
// Inicialização
Para todos os nodos j
     dist(j)=INF    // INF constante muito elevada
     pred(j)=0      // não há predecessores
dist(1)=0
L={1}
// Ciclo
Enquanto L não estiver vazio
     Remover um nodo i do conjunto L
     Para todos os arcos ij
          Se dist(j)>dist(i)+custo(ij)
               dist(j)=dist(i)+custo(ij)
               se dist(j)<nC    // nC limite inferior para dist
                    sair // detectada a existência de ciclo negativo
               se não
                    pred(j)=i
                    adicionar j ao conjunto L
```

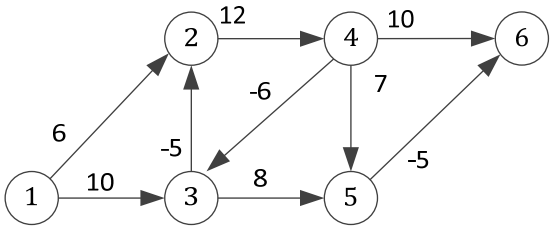


Figura 3.16. Exemplo de rede com custos negativos.

Iteração	L	i	Nodo (dist,pred)					
			1	2	3	4	5	6
0	-	-	(0,0)	(INF,0)	(INF,0)	(INF,0)	(INF,0)	(INF,0)
1	{1}	1		(6,1)	(10,1)			
2	{2,3}	2				(18,2)		
3	{3,4}	3		(5,3)			(18,3)	
4	{4,2,5}	4						(28,4)
5	{2,5,6}	2				(17,2)		
6	{5,6,4}	5						(13,5)
7	{6,4}	6						
8	{4}	4						
9	{}							

Tabela 3.3. Aplicação do algoritmo de Bellman-Ford.

3.4 *Caminhos mais curtos entre todos os pares de nodos

3.4.1 Programação Inteira

Um modelo de programação inteira para o problema de determinar os caminhos mais curtos entre todos os pares de nodos parte da definição do conjunto de pares de nodos $K = \{(1,2),(1,3),\dots,(n-1,n)\}$. Cada elemento de K, conjunto indexado por k, tem um nodo origem (o nodo inicial do caminho) o^k e um nodo destino (o nodo final do caminho), d^k, em que $o^k \neq d^k$.

Define-se as seguintes variáveis de decisão:

$$x_{ij}^k = \begin{cases} 1 \; se \; arco \; ij \; faz \; parte \; do \; caminho \\ \quad mais \; curto \; do \; par \; k \\ 0 \; caso \; contrário \end{cases}, \forall ij \in A, \forall k \in K$$

Modelo de programação inteira:

$$Min\, z = \sum_{k \in K} \sum_{ij \in A} c_{ij}\, x_{ij}^k$$

$$sujeito\ a:$$

$$\sum_{j:ij \in A} x_{ij}^k - \sum_{j:ji \in A} x_{ji}^k = \begin{cases} 1, \; se \; i = o^k \\ 0, \; se \; i \neq o^k, i \neq d^k \\ -1, se \; i = d^k \end{cases}, \forall i \in N, \forall k \in K$$

$$x_{ij}^k \in \{0,1\}, \forall ij \in A, \forall k \in K$$

3.4.2 Algoritmo específico

O algoritmo de Floyd-Warshall permite determinar o caminho mais curto entre todos os pares de nodos de forma mais eficiente do que determinar o caminho mais curto para todos os pares de nodos de forma independente ou determinar o caminho mais curto entre cada nodo e todos os restantes para todos os nodos.

O algoritmo utiliza duas matrizes: uma matriz $dist(i,j)$ que guarda a menor distância obtida até ao momento entre os nodos i e j e uma matriz $pred(i,j)$ que guarda o nodo predecessor de j no caminho de menor distância entre i e j encontrado até ao momento. Em cada iteração é tentada a actualização do caminho mais curto entre cada par de nodos por utilização de um dos nodos.

```
// Algoritmo de Floyd-Warshall para o problema do caminho mais curto
          entre um nodo e todos os restantes em redes com custos
          arbitrários
// Inicialização
Para todos os pares de nodos ij
     dist(i,j)=INF // INF constante muito elevada
     pred(i,j)=0    // não há predecessores
Para todos os nodos i
     dist(i,i)=0
Para todos os arcos ij
     dist(i,j)=custo(i,j)
     pred(i,j)=i
// Ciclo
Para k=1 até n
     Para todos os pares de nodos ij
          Se dist(i,j)>dist(i,k)+ dist(k,j)
               dist(i,j)=dist(i,k)+ dist(k,j)
               pred(i,j)=pred(k,j)
               se dist(i,j)<nC  // nC limite inferior para dist
                    sair // detectada a existência de ciclo negativo
```

3.5 Extensões e variantes

3.5.1 Restrições de recursos

Num problema de caminhos mais curto com restrições de recursos existe, associado a cada arco, para além do seu custo, o consumo de (pelo menos) um recurso. Consideram-se agora dois modelos de programação inteira para dois casos particulares relevantes: o problema do caminho mais curto com um único recurso e o problema do caminho mais curto restringido pelo número de saltos (que por sua vez é também um caso particular do problema do caminho mais curto com um recurso). A extensão para o caso geral, que compreende vários recursos, é directa.

Recurso único

Considere-se uma rede em que a cada arco está associado um custo, c_{ij}, e uma duração, t_{ij}, $\forall ij \in A$. Pretende-se determinar o caminho mais curto (relativamente aos custos nos arcos como anteriormente) mas cuja duração total não excede uma constante b.

A título exemplificativo, considere-se a rede da Figura 3.17. Junto a cada arco é dado o seu custo e a sua duração, por esta ordem. Pretende-se determinar o caminho mais curto entre os nodos 1 e 6 cuja duração que não ultrapasse 14 unidades de tempo.

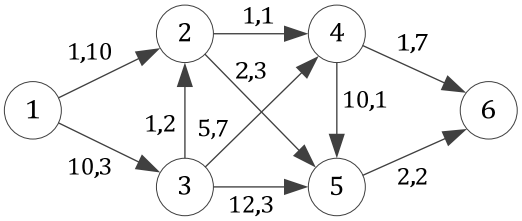


Figura 3.17. Rede de um problema de caminho mais curto com um recurso.

Um modelo de programação inteira geral é:

$$Min\ z = \sum_{ij \in A} c_{ij} x_{ij}$$

$$sujeito\ a:$$

$$\sum_{j:ij \in A} x_{ij} - \sum_{j:ji \in A} x_{ji} = \begin{cases} 1, se\ i = o \\ 0, se\ i \neq d, i \neq d, \forall i \in N \\ -1, se\ i = d \end{cases}$$

$$\sum_{ij \in A} t_{ij} x_{ij} \leq b$$

$$x_{ij} \in \{0,1\}, \forall ij \in A$$

De forma geral, os problemas de caminho mais curto com restrições de recursos não têm a propriedade da integralidade, pelo que não é possível relaxar as restrições de integralidade de x_{ij} (isto é, a substituição de $x_{ij} \in \{0,1\}, \forall ij \in A$ por $0 \leq x_{ij} \leq 1, \forall ij \in A$, pode conduzir a soluções fraccionárias).

Número de saltos

O número de saltos de um caminho é o seu número de arcos (ou o seu número de nodos menos um). O problema do caminho mais curto com a restrição de o seu número de saltos ter de ser menor ou igual a uma constante b.

Um modelo de programação inteira geral obtém-se do modelo para o problema de recurso único definindo $t_{ij} = 1$:

$$Min\ z = \sum_{ij \in A} c_{ij} x_{ij}$$

$$sujeito\ a:$$

$$\sum_{j:ij \in A} x_{ij} - \sum_{j:ji \in A} x_{ji} = \begin{cases} 1, se\ i = o \\ 0, se\ i \neq d, i \neq d, \forall i \in N \\ -1, se\ i = d \end{cases}$$

$$\sum_{ij \in A} x_{ij} \leq b$$

$$x_{ij} \in \{0,1\}, \forall ij \in A$$

Em (Pioro and Medhi 2004) é apresentado um algoritmo específico para o problema do caminho mais curto com uma restrição de número de saltos, aqui apresentou-se um modelo de programação inteira, salientando a facilidade em integrar componentes deste modelo noutros

modelos mais elaborados (por exemplo, num modelo para determinar caminhos mais curtos disjuntos com uma restrição do número de saltos).

Caso geral

Considere-se agora que existem R recursos, a inclusão de um arco no caminho corresponde ao consumo de t_{ij}^{r} unidades do recurso r, $r \in R$, e a disponibilidade do recurso r é $b_r, r \in R$. O modelo de programação inteira é:

$$Min\ z = \sum_{ij \in A} c_{ij} x_{ij}$$

$$sujeito\ a:$$

$$\sum_{j:ij \in A} x_{ij} - \sum_{j:ji \in A} x_{ji} = \begin{cases} 1, se\ i = o \\ 0, se\ i \neq d, i \neq d \\ -1, se\ i = d \end{cases}, \forall i \in N$$

$$\sum_{ij \in A} t_{ij}^{r} x_{ij} \leq b_{r_r}, \forall r \in R$$

$$x_{ij} \in \{0,1\}, \forall ij \in A$$

3.5.2 Caminhos disjuntos

Caminhos disjuntos nos arcos

Considera-se o problema de determinar dois caminhos disjuntos nos arcos (cada arco não pode fazer parte de mais de um caminho) entre dois nodos de forma a minimizar o custo total (soma dos custos dos arcos dos dois caminhos).

A solução deste problema não pode ser obtida, calculando primeiro o caminho mais curto e depois o segundo caminho mais curto que não tenha arcos do primeiro. Tal é ilustrado na Figura 3.18. Os dois caminhos mais curtos disjuntos são 1-2-4 e 1-3-4 com custo 22. O caminho mais curto é 1-2-3-4 e, não usando nenhum arco deste caminho para se obter um caminho disjunto, obtém-se o segundo caminho 1-4, sendo o custo total 103.

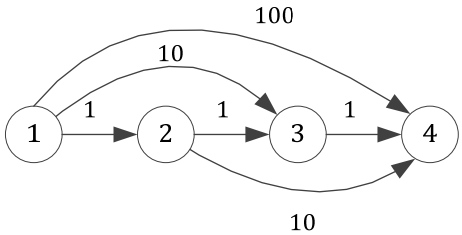


Figura 3.18. Ilustração da diferença do problema dos dois caminhos disjuntos com menor custo total e do problema dos dois caminhos mais curtos.

Um modelo de programação inteira é o seguinte.

Variáveis de decisão:

$$x_{ij}^{1} = \begin{cases} 1\ se\ arco\ ij\ faz\ parte\ de\ um\ dos\ dois\ caminhos \\ 0\ caso\ contrário \end{cases}, \forall ij \in A$$

$$x_{ij}^2 = \begin{cases} 1\ se\ arco\ ij\ faz\ parte\ do\ outro\ caminho \\ 0\ caso\ contrário \end{cases}, \forall ij \in A$$

Modelo de programação inteira:

$$Min\ z = \sum_{k=1}^{2} \sum_{ij \in A} x_{ij}^k$$

$$sujeito\ a:$$

$$\sum_{j:ij \in A} x_{ij}^k - \sum_{i:ji \in A} x_{ij}^k = \begin{cases} 1,\ se\ i = o \\ 0,\ se\ i \neq o, i \neq d \\ -1, se\ i = d \end{cases}, \forall i \in N,\ k = 1,2$$

$$\sum_{k=1}^{2} x_{ij}^k \leq 1,\ \forall ij \in A$$

$$x_{ij}^k \in \{0,1\}, \forall ij \in A,\ k = 1,2$$

A extensão para o caso geral em que se pretende p caminhos disjuntos nos arcos é directa.

Caminhos disjuntos nos nodos

Dois caminhos serem disjuntos nos arcos não garante que sejam disjuntos nos nodos (já dois caminhos serem disjuntos nos nodos é condição suficiente para serem disjuntos nos arcos).

Um modelo de programação inteira para o problema dos dois caminhos disjuntos nos nodos obtém-se substituindo o último grupo de restrições do modelo dos dois caminhos disjuntos nos arcos (exceptuando as do domínio das variáveis) por

$$\sum_{j:ij \in A} \sum_{k=1}^{2} x_{ij}^k \leq 1,\ \forall i \in N \setminus \{o\}$$

Estas restrições forçam a que, de cada nodo (excepto o nodo inicial do caminho), saia no máximo um arco que não pode pertencer a ambos os caminhos.

Por exemplo, para a rede da Figura 3.18, a restrição do nodo 2 fica:

$$x_{23}^1 + x_{24}^1 + x_{23}^2 + x_{24}^2 \leq 1$$

Desta forma, o nodo 2 só pode ser visitado por um dos dois caminhos.

A extensão para o caso geral em que se pretende p caminhos disjuntos nos nodos é directa.

3.5.3 Objectivo minmax

Ao contrário dos problemas abordados até agora, no problema com objectivo minmax o valor de um caminho não corresponde à soma dos custos dos arcos que o compõem mas sim ao maior custo de entre os custos dos arcos que o compõem. O valor de um caminho P é assim dado por $\max_{ij \in P}\{c_{ij}\}$. Por exemplo, o caminho $1-2-5-8-10$ da Figura 3.19 tem o valor 7. Pretende-se o caminho cujo maior custo é o menor possível.

Uma solução óptima para a instância da Figura 3.19 é o caminho $1-4-6-9-10$ com valor 4.

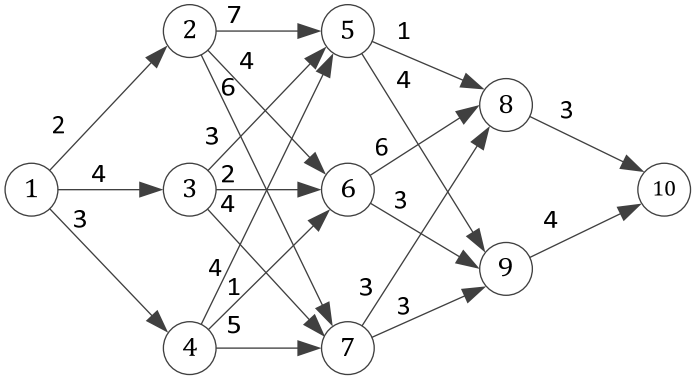


Figura 3.19. Rede do exemplo do caminho mais curto com objectivo minmax.

Um modelo de programação inteira é agora apresentado.

Variáveis de decisão:

$$z - custo\ do\ arco\ com\ maior\ custo\ do\ caminho\ seleccionado$$

$$x_{ij} = \begin{cases} 1\ se\ arco\ ij\ faz\ parte\ do\ caminho\ mais\ curto \\ 0\ caso\ contrário \end{cases}, \forall ij \in A$$

Modelo de programação inteira:

$$Min\ z$$

$$sujeito\ a:$$

$$\sum_{j:ij \in A} x_{ij} - \sum_{j:ji \in A} x_{ji} = \begin{cases} 1,\ se\ i = o \\ 0,\ se\ i \neq s, i \neq t \\ -1, se\ i = d \end{cases}, \forall i \in N$$

$$z \geq c_{ij} x_{ij}, \forall ij \in A$$

$$z \geq 0$$

$$x_{ij} \in \{0,1\}, \forall ij \in A$$

3.5.4 Caminho mais fiável

Considere-se um problema em que se pretende obter o caminho mais fiável numa rede em que são conhecidas as probabilidades dos arcos não falharem. A probabilidade do arco ij não falhar (i.e. é a fiabilidade do arco ij) é dada por p_{ij}.

A probabilidade de um caminho P não falhar é dada pelo produto da fiabilidade dos arcos que o compõem. Assim, o problema de determinar o caminho mais fiável pode ser modelado através de programação inteira

$$Max \prod_{ij \in A} p_{ij} x_{ij}$$

$$sujeito\ a:$$

$$\sum_{j:ij \in A} x_{ij} - \sum_{j:ji \in A} x_{ji} = \begin{cases} 1,\ se\ i = o \\ 0,\ se\ i \neq s, i \neq t \\ -1, se\ i = d \end{cases}, \forall i \in N$$

$$x_{ij} \in \{0,1\}, \forall ij \in A$$

A função objectivo não é linear o que dificulta em muito a resolução do problema. No entanto, é possível efectuar uma transformação para um problema equivalente, tendo em conta que um caminho que maximize a função objectivo original (fiabilidade) maximiza a função

$$Max\ log \prod_{ij \in A} p_{ij} x_{ij}$$

que é equivalente a

$$Min\ -log \prod_{ij \in A} p_{ij} x_{ij}$$

que é equivalente a

$$Min \sum_{ij \in A} (-log\, p_{ij}) x_{ij}$$

o que transforma o problema original (caminho mais fiável) no problema de caminho mais curto:

$$Min \sum_{ij \in A} (-log\, p_{ij}) x_{ij}$$

$$sujeito\ a:$$

$$\sum_{j:ij \in A} x_{ij} - \sum_{j:ji \in A} x_{ji} = \begin{cases} 1,\ se\ i = o \\ 0,\ se\ i \neq s, i \neq t\ , \forall i \in N \\ -1, se\ i = d \end{cases}$$

$$x_{ij} \in \{0,1\}, \forall ij \in A$$

Como exemplo, considere-se a rede da Figura 3.20.

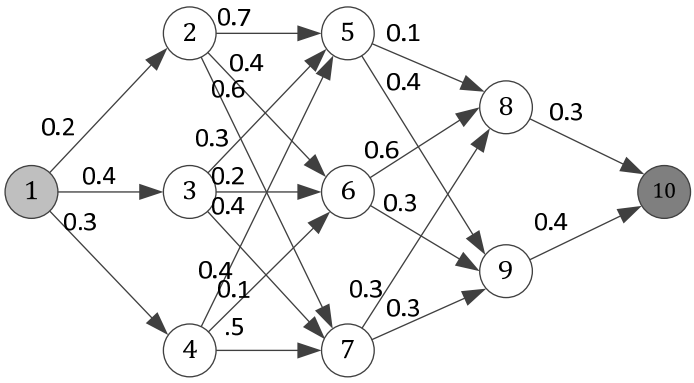


Figura 3.20. Instância do problema do caminho mais fiável (a cinzento claro o nodo inicial e a cinzento escuro o nodo final).

O coeficiente de uma variável x_{ij} no problema do caminho mais curto é dado por $-\log p_{ij}$, por exemplo o coeficiente do arco 12 é $-\log 0.2 = 0.70$. A solução óptima do problema de caminho mais curto tem valor 1.65 e corresponde ao caminho 1-2-5-9-10 com fiabilidade 0.0224.

3.5.5 *k* caminhos mais curtos

No problema dos *k* caminhos mais curtos, pretende-se determinar um determinado número *k* (por exemplo, 5) de caminhos mais curtos entre dois nodos. Embora haja algoritmos específicos para este problema (ver, por exemplo, Queirós et al. 1999 ou Pioro and Medhi 2004), será aqui apresentada apenas uma abordagem baseada em programação inteira.

Formular um modelo de programação inteira para o problema dos k caminhos mais curtos não é fácil. Opta-se por uma abordagem em que o modelo do caminho mais curto entre dois nodos é modificado e resolvido sucessivamente. Na primeira iteração, obtém-se o caminho mais curto, na segunda, o segundo caminho mais curto, e assim sucessivamente até à iteração k.

O que se altera de uma iteração para a outra é a inclusão de uma nova restrição no modelo actual.

Em geral, a segunda melhor solução óptima de um modelo de programação inteira pode ser obtida da seguinte forma. Considere-se que x^* é a solução óptima actual e que as variáveis com valor 1 nessa solução definem o subconjunto P do conjunto de todas as variáveis A. Para se obter uma solução diferente da actual, pelos menos uma variável tem de mudar de valor:

$$\sum_{ij \in P} (1 - x_{ij}) + \sum_{ij \in A \setminus P} x_{ij} \geq 1$$

O primeiro somatório corresponde a um número de variáveis que passam de 1 para 0. O segundo somatório corresponde a um número de variáveis que passam de 0 para 1. A soma dos dois tem de ser maior ou igual a um já que se pretende uma solução diferente, i.e. que a solução actual não seja admissível.

Após manipulação algébrica a expressão obtém-se a restrição

$$\sum_{ij \in P} x_{ij} - \sum_{ij \in A \setminus P} x_{ij} \leq |P| - 1$$

No caso concreto em que se pretende evitar um caminho, é possível simplificar a restrição para

$$\sum_{ij \in P} x_{ij} \leq |P| - 1$$

o que corresponde a forçar a que pelo menos um arco do caminho deixe de fazer parte da solução e um caminho diferente seja encontrado.

Alternativamente a restrição a considerar pode ser

$$\sum_{ij \in A \setminus P} x_{ij} \geq 1$$

obrigando a que pelo menos um arco que não fazia parte do caminho passe a fazer parte. Esta forma é menos preferível do que a anterior por ser uma restrição mais densa – com mais variáveis.

A solução óptima da instância do problema do caminho mais curto entre 1 e 6 da rede da Figura 3.8 é $x_{13} = x_{35} = x_{56} = 1$ (e as restantes variáveis com valor zero) com valor 9. A restrição a introduzir no modelo para obter o segundo caminho mais curto é assim:

$x_{13} + x_{35} + x_{56} \leq 2$.

A solução obtida é $x_{13} = x_{34} = x_{46} = 1$ com valor 12.

Optimiza-se agora o modelo com as restrições originais e

$$x_{13} + x_{35} + x_{56} \leq 2$$
$$x_{13} + x_{34} + x_{46} \leq 2$$

obtendo-se o terceiro caminho mais curto $x_{13} = x_{35} = x_{54} = x_{45} = 1$ com valor 16.

Note-se que, quando se inserem restrições, o modelo do caminho mais curto perde a propriedade da integralidade (a solução óptima não é necessariamente inteira) pelo que é necessário forçar as variáveis a tomarem valores binários.

3.6 Caminho preferido bi-objectivo

3.6.1 Definição do problema

O problema do caminho preferido bi-objectivo é um problema em que se pretende escolher um caminho de acordo com dois objectivos, por exemplo, o custo e a duração. Um modelo bi-objectivo é

$$Min\ z_1 = \sum_{ij \in A} c_{ij} x_{ij}$$

$$Min\ z_2 = \sum_{ij \in A} t_{ij} x_{ij}$$

$$sujeito\ a:$$

$$\sum_{j:ij \in A} x_{ij} - \sum_{j:ji \in A} x_{ji} = \begin{cases} 1, se\ i = o \\ 0, se\ i \neq d, i \neq d, \forall i \in N \\ -1, se\ i = d \end{cases}$$

$$x_{ij} \in \{0,1\}, \forall ij \in A$$

em que a cada arco está associado um custo, c_{ij}, e uma duração, t_{ij}, $\forall ij \in A$.

Tipicamente, os dois objectivos são conflituosos e portanto a escolha de um caminho envolve a participação directa ou através do estabelecimento de preferências do agente de decisão.

Considera-se o exemplo dado na Figura 3.17. em que junto a cada arco é dado o seu custo (em €) e a sua duração (em horas), por esta ordem. As duas funções objectivo são

$Min\ z_1 = x_{12} + 10\ x_{13} + x_{24} + 2\ x_{25} + x_{32} + 5\ x_{34} + 12\ x_{35} + 10\ x_{45} + x_{46} + 2\ x_{56}$

$Min\ z_2 = 10\ x_{12} + 3\ x_{13} + x_{24} + 3\ x_{25} + 2\ x_{32} + 7\ x_{34} + 3\ x_{35} + x_{45} + 7\ x_{46} + 2\ x_{56}$

A solução que minimiza o custo é o caminho 1-2-4-6 com custo 3 € (e duração 18 horas). A solução que minimiza a duração é o caminho 1-3-5-6 com duração 8 horas (e custo 24 €).

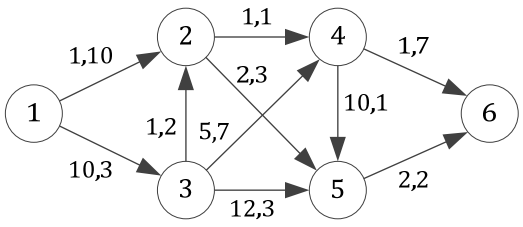


Figura 3.21. Rede do exemplo do caminho preferido b-objectivo.

Apenas com intervenção do agente de decisão é possível escolher uma solução entre as duas obtidas ou eventualmente outra que também seja não dominada. Uma solução é

dominada se existir outra com menor custo e menor duração (ou menor custo e igual duração ou menor duração e igual custo). Uma solução não dominada é uma solução eficiente. Racionalmente, o agente de decisão deverá preferir uma solução eficiente.

A optimização tem como papel identificar uma solução preferida de acordo com informação prestada pelo agente de decisão ou um (sub)conjunto de soluções eficientes para posterior análise por parte do agente de decisão.

3.6.2 Identificação de soluções eficientes

Em problemas em que é possível caracterizar todas as soluções, é possível reduzir o número de soluções candidatas a solução preferida ao excluir as soluções dominadas (um solução é dominada por outra se não é melhor que ela em nenhum objectivo).

Considere-se um problema em que se pretende instalar um cabo entre dois pontos. Existem seis caminhos alternativos, cada um deles envolvendo um determinado comprimento e um determinado custo (que se pretendem minimizar), valores dados na Tabela 3.4 e representados na Figura 3.22.

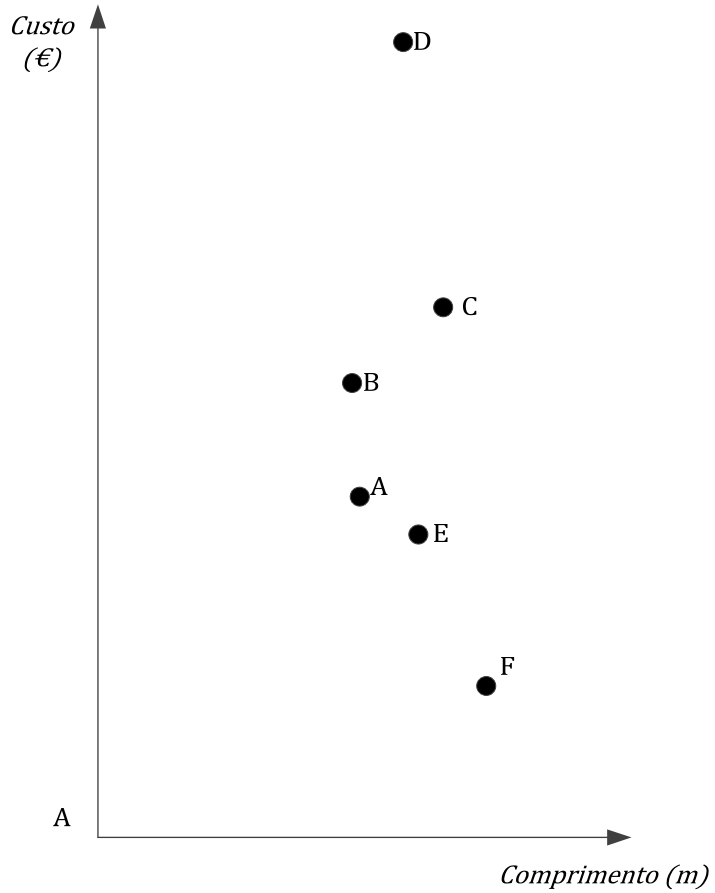


Figura 3.22. Gráfico das soluções e valores em cada atributo.

	Comprimento (m)	Custo (€)
A	687	900
B	667	1200
C	906	1400
D	801	2100
E	841	800
F	1019	400

Tabela 3.4. Valor em cada atributo de cada caminho.

As soluções C e D são dominadas (C por E, A e B; D por A e B). As soluções eficientes são a A, a B, a E e a F.

3.6.3 Método de agregação por pesos

No método de agregação por pesos o agente de decisão indica a importância relativa das funções objectivo através de pesos, permitindo assim que o problema bi-objectivo original seja transformado num problema com apenas um objectivo;

$$Min\ z = \lambda_1 \sum_{ij \in A} c_{ij} x_{ij} + \lambda_2 \sum_{ij \in A} t_{ij} x_{ij}$$

$$sujeito\ a:$$

$$\sum_{j:ij \in A} x_{ij} - \sum_{j:ji \in A} x_{ji} = \begin{cases} 1, se\ i = o \\ 0, se\ i \neq d, i \neq d \\ -1, se\ i = d \end{cases}, \forall i \in N$$

$$x_{ij} \in \{0,1\}, \forall ij \in A$$

Após a atribuição dos pesos, o problema resultante é optimizado (em multi-objectivo) ou é calculado o valor agregado de cada solução (em multi-atributo) e a solução óptima, aquela com menor valor agregado (em minimização), é a solução preferida.

A título exemplificativo, considere-se que o agente de decisão considera que a duração é 3 vezes mais importante do que o custo. Tal significa resolver o problema agregado com $\lambda_1 = 1$ e $\lambda_2 = 3$ que tem como solução óptima o caminho 1-3-2-5-6 (custo 15 €, duração 10 horas).

Os pesos podem ser vistos como a definição de um compromisso (*trade-off*) que estabelece quanto o agente de decisão está disposto a piorar num objectivo para melhorar noutro. No exemplo, o agente de decisão está disposto a pagar mais 3 € pela redução de 1 hora. Em geral, o agente de decisão está disposto a pagar mais λ_2 € por reduzir a duração em λ_1 horas.

A utilização deste método pressupõe o mesmo compromisso quaisquer que sejam os valores das funções objectivo, ficando a relação entre os pesos completamente definida pela razão λ_2/λ_1.

Existem métodos relativamente elaborados para auxiliar a determinação de um compromisso, nomeadamente perguntando explicitamente ao agente de decisão a sua preferência (ou indiferença) entre duas soluções e, com a informação obtida, refazer a questão com outras soluções.

O método de agregação por pesos é particularmente atractivo porque transforma um problema multi-objectivo (ou multi-atributo) num problema de objectivo único, simplificando-o significativamente. No entanto, a sua aplicação deve ser cuidadosa por três motivos.

O primeiro é ser particularmente difícil o agente de decisão definir um valor de compromisso entre os objectivos. Acresce que o compromisso pode não ser o mesmo para todos os valores dos objectivos. No exemplo, o agente de decisão pode estar disposto a pagar 3 € por hora para durações pequenas mas apenas 1 € por hora para durações maiores. Em geral, a função que estabelece os compromissos do agente de decisão (função utilidade) pode ser difícil de obter e de tratar analiticamente.

O segundo motivo está relacionado com pequenas variações dos pesos poderem conduzir a soluções óptimas do problema agregado muito diferentes.

O terceiro motivo é que há soluções eficientes que, para qualquer escolha de pesos, nunca são preferidas.

3.6.4 Método de geração através de pesos

No método de geração através de pesos é obtido um conjunto de soluções que o agente de decisão analisará e de entre as quais escolherá a solução preferida.

Este método usa diferentes pesos para gerar soluções eficientes. Aqui os pesos são parâmetros, não estando relacionados com julgamentos do agente de decisão sobre a importância relativa dos diferentes objectivos. Tipicamente, o número de soluções eficientes é extremamente elevado, sendo pouco razoável almejar obter toda a fronteira eficiente.

No método de geração por pesos, é aconselhável recorrer à normalização de escalas dos objectivos (nomeadamente quando as escalas têm magnitudes muito diferentes). Um ponto com valor z^{orig} deve passar a valer

$$z^{norm} = \frac{z^{orig} - z^{min}}{z^{max} - z^{min}}$$

em que z^{max} e z^{min} são o maior e menor valores, respectivamente, que a função objectivo em causa podem tomar, em geral obtidos (no caso dos mínimos) ou estimados (no caso dos máximos) considerando a optimização isolada de cada objectivo.

No exemplo, $z_1^{max} = 24$, $z_2^{max} = 18$, $z_1^{min} = 3$ e $z_2^{min} = 8$.

As funções objectivo passam a

$$Min\ z_1^n = \frac{x_{12}+10\,x_{13}+x_{24}+2\,x_{25}+x_{32}+5\,x_{34}+12\,x_{35}+10\,x_{45}+x_{46}+2\,x_{56}-3}{24-3}$$

$$Min\ z_2^n = \frac{10\,x_{12}+3\,x_{13}+x_{24}+3\,x_{25}+2\,x_{32}+7\,x_{34}+3\,x_{35}+x_{45}+7\,x_{46}+2\,x_{56}-8}{18-8}$$

Note-se que as constantes não influenciam a optimização.

Pretendendo-se um conjunto de soluções eficientes diverso, os pesos devem respeitar

$$\lambda_1 + \lambda_2 = 1$$
$$\lambda_1, \lambda_2 \geq 0$$

o que significa que a definição de um peso é suficiente ($\lambda_2 = 1 - \lambda_1$).

Tomando o valor de q como o número de optimizações da função agregada a realizar (um limite superior para o número de soluções eficientes a gerar), o valor inicial para λ_1 deverá ser 0 e deverá ser incrementado de $\frac{1}{(q-1)}$ a cada optimização atingindo na última o valor 1.

No exemplo, para 10 optimizações, obtêm-se as soluções mostradas na Tabela 3.5. Assim, com este método obtiveram-se 4 soluções eficientes.

λ_1	$\lambda_2 = 1 - \lambda_1$	Custo	Duração	Caminho
0.00	1.00	24	8	1-3-5-6
0.11	0.89	24	8	1-3-5-6
0.22	0.78	24	8	1-3-5-6
0.33	0.67	15	10	1-3-2-5-6
0.44	0.56	15	10	1-3-2-5-6
0.55	0.45	5	15	1-2-5-6
0.66	0.34	5	15	1-2-5-6
0.77	0.23	3	18	1-2-4-6
0.88	0.12	3	18	1-2-4-6
0.99	0.01	3	18	1-2-4-6

Tabela 3.5. Soluções eficientes obtidas pelo método de geração através de pesos.

3.6.5 Distância ao ideal

A solução ideal é uma solução (usualmente não admissível) que tem o valor óptimo em cada um dos objectivos quando considerados individualmente.

A participação do agente de decisão neste método é muito reduzida podendo traduzir-se apenas na escolha de uma métrica (forma de quantificar a distância) e, eventualmente, na uniformização das escalas e/ou normalização dos objectivos.

Uma métrica que possibilita a linearidade dos modelos é a métrica de Manhattan (ou de Hamming). Nesta métrica a distância entre dois pontos é a soma dos valores absolutos das diferenças entre as suas coordenadas.

No exemplo, o problema de determinar a distância ao ideal (em maximização) é:

$$Min\ s_1 + s_2$$

$$sujeito\ a:$$

$$x_{12} + 10\,x_{13} + x_{24} + 2\,x_{25} + x_{32} + 5\,x_{34} + 12\,x_{35} + 10\,x_{45} + x_{46} + 2\,x_{56} - s_1 = 3$$

$$10\,x_{12} + 3\,x_{13} + x_{24} + 3\,x_{25} + 2\,x_{32} + 7\,x_{34} + 3\,x_{35} + x_{45} + 7\,x_{46} + 2\,x_{56} - s_2 = 8$$

$$\sum_{j:ij\in A} x_{ij} - \sum_{j:ji\in A} x_{ji} = \begin{cases} 1, se\ i = o \\ 0, se\ i \neq d, i \neq d \\ -1, se\ i = d \end{cases}, \forall i \in N$$

$$x_{ij} \in \{0,1\}, \forall ij \in A$$

$$s_1, s_2 \geq 0$$

As variáveis de decisão s medem a distância (de Manhattan) à solução ideal (que tem custo 3 e duração 8). Note-se que a escala das duas funções objectivo é determinante na solução obtida pelo que a normalização e / ou a participação do agente de decisão na definição de um compromisso para as distâncias ao ideal (variáveis s) conduzirão a melhores resultados.

3.6.6 Optimização lexicográfica

A optimização lexicográfica é utilizada quando existe uma clara hierarquia entre objectivos definida pelo agente de decisão.

Considerando que o índice dos objectivos está de acordo com essa hierarquia (o objectivo mais importante é o f_1), o que a hierarquização dos objectivos estabelece é que uma solução com melhor valor do que outra no primeiro objectivo é preferível a essa outra quaisquer que sejam os valores de ambas nos restantes objectivos. Se duas soluções tiverem o mesmo valor no primeiro objectivo, o raciocínio anterior aplica-se ao segundo objectivo e assim sucessivamente.

O primeiro problema a resolver é:

$$Min\ z_1 = f_1(x)$$
$$sujeito\ a:$$
$$x \in X$$

Representando por z_1^* o valor óptimo deste problema, de seguida será resolvido o problema que corresponde a obter o valor óptimo das soluções que minimizam o segundo objectivo e são óptimas (alternativas) do primeiro

$$Min\ z_2 = f_2(x)$$
$$sujeito\ a:$$
$$x \in X$$
$$f(x) = z_1^*$$

O k-ésimo problema corresponde a obter o valor óptimo das soluções que minimizam o k-ésimo objectivo e são óptimas (alternativas) de todos os anteriores

$$Min\ z_k = f_k(x)$$
$$sujeito\ a:$$
$$x \in X$$
$$f_j(x) = z_j^*, j = 1, \dots, k-1$$

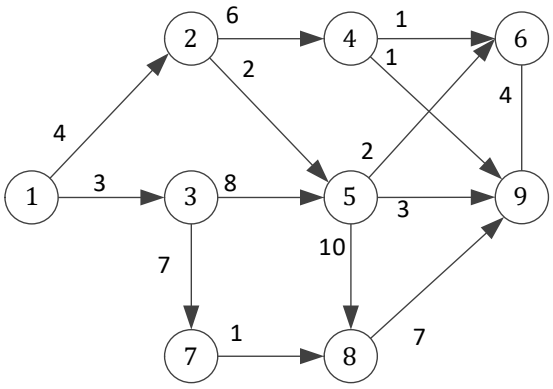
No exemplo dado na Figura 3.17., suponha-se que se pretende minimizar hierarquicamente o número de saltos e depois o custo.

A primeira optimização resulta num número mínimo de saltos igual a 3. Adicionando a restrição que o caminho tem de ter três arcos ao problema da minimização do custo, obtém-se a solução óptima lexicográfica.

3.7 Exercícios

Exercício 3.1 Caminho mais curto (modelo + *solver*)

Considere a seguinte rede.



a) Apresente um modelo de programação inteira que lhe permita determinar o caminho mais curto entre os nodos 1 e 9.

b) Obtenha esse caminho através da utilização de software.

Exercício 3.2 Caminho mais curto (dimensão do modelo)

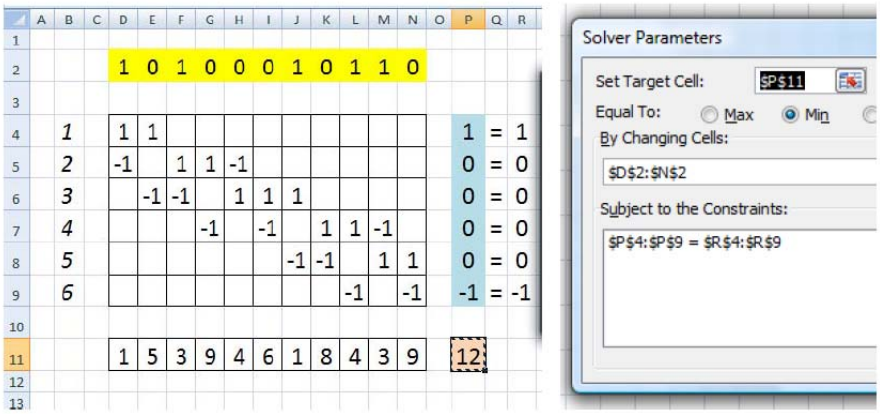
Indique o número de variáveis de decisão, o número de restrições e o número de coeficientes não nulos das variáveis de decisão nas restrições de um modelo de programação linear para o problema do caminho mais curto entre dois nodos definido numa rede com n nodos e m arcos.

Exercício 3.3 Caminho mais curto (rede + *solver*)

A folha de cálculo seguinte foi construída para resolver um problema de caminho mais curto com um solver de programação inteira.

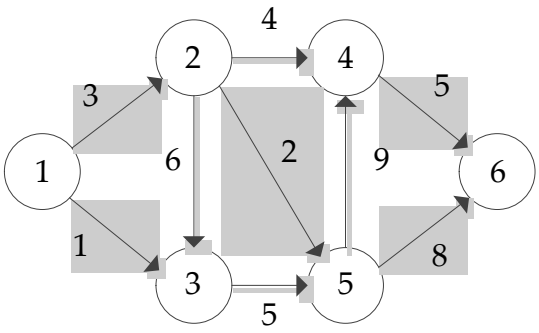
a) Represente a rede correspondente e a solução indicada na folha de cálculo.

b) Quais as fórmulas que foram inseridas nas células P4 a P9? E na célula P11?



Exercício 3.4 Caminho mais curto (modelo + *solver*)

Considere a rede da figura.

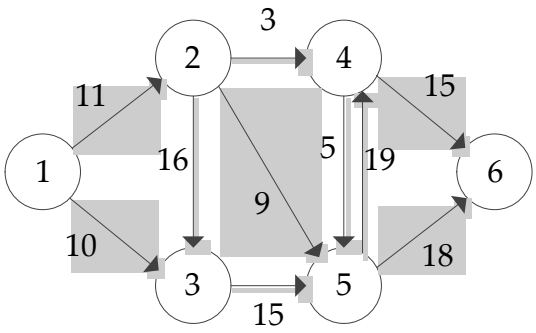


a) Apresente o modelo de programação linear para o problema do caminho mais curto entre 1 e 6.

b) Indique como resolveria o problema com o *solver* de uma folha de cálculo.

Exercício 3.5 Árvore dos caminhos mais curtos (modelo + *solver*)

Considere a rede da figura.



a) Apresente um modelo desagregado e um modelo agregado para o problema da árvore dos caminhos mais curtos com raiz em 1.

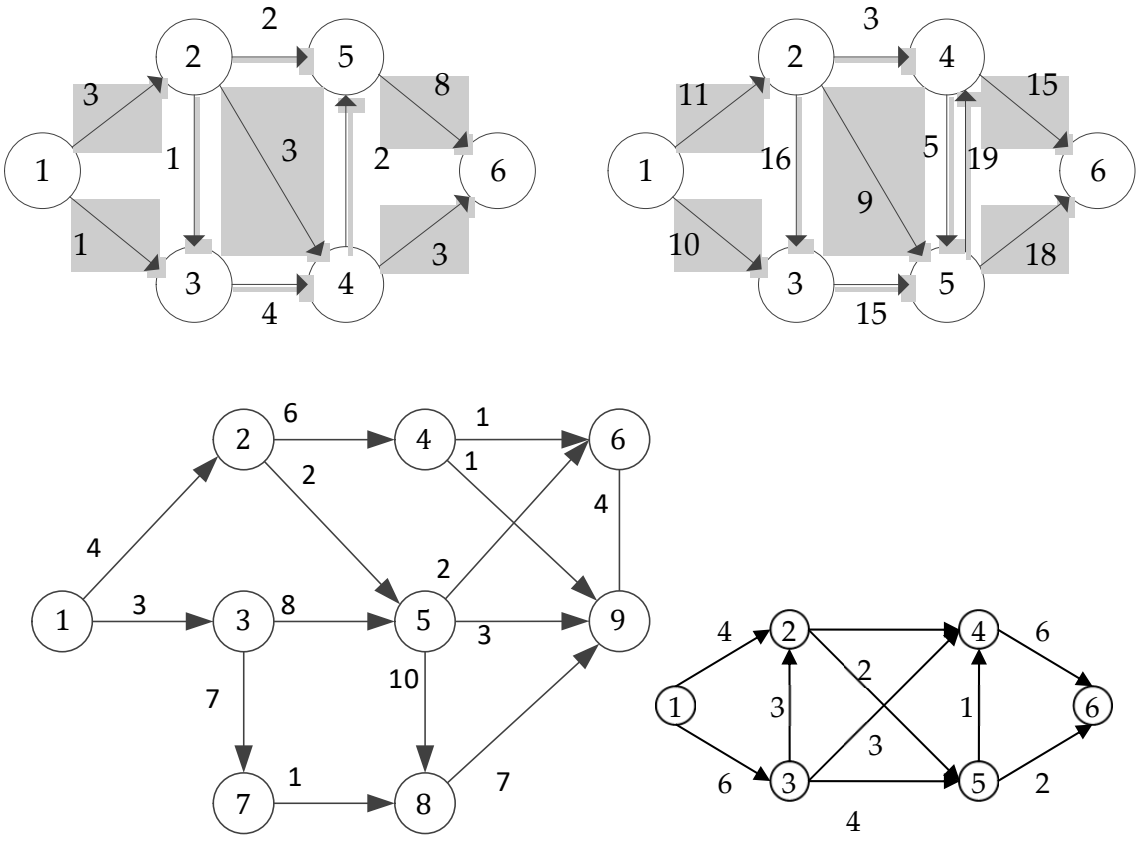
b) Obtenha a solução óptima de ambos os modelos com utilização de software.

Exercício 3.6 Árvore dos caminhos mais curtos (algoritmos específicos)

Para cada uma das quatro redes seguintes.

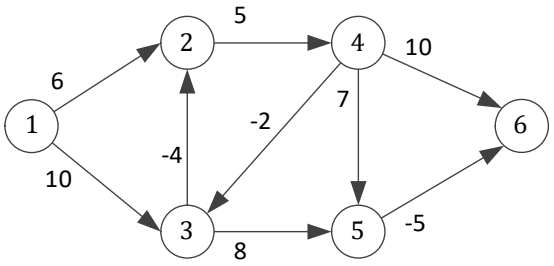
a) Mostre, através da aplicação de um algoritmo, que a rede é cíclica ou acíclica.

b) Indique qual o algoritmo adequado para resolver o problema árvore dos caminhos mais curtos com raiz em 1 e aplique-o.



Exercício 3.7 * Caminho mais curto (algoritmo específico + modelo + *solver*)

Considere a rede da figura.



a) Por inspecção, obtenha uma solução óptima do problema do caminho mais curto (elementar) entre os nodos 1 e 6.

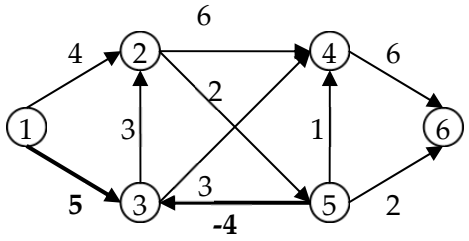
b) Indique a solução retornada pelo modelo de programação inteira discutido neste capítulo. A solução é admissível?

c) Confirme, com a utilização de software, as soluções que obteve nas duas alíneas anteriores. No caso da alínea b), utilize o resultado de um ciclo poder ser eliminado do conjunto das soluções admissíveis através da restrição $\sum_{ij \in C} x_{ij} \leq |C|$ em que C é o conjunto dos arcos do ciclo e $|C|$ o número de arcos do ciclo C.

d) Aplique o algoritmo de Bellman-Ford.

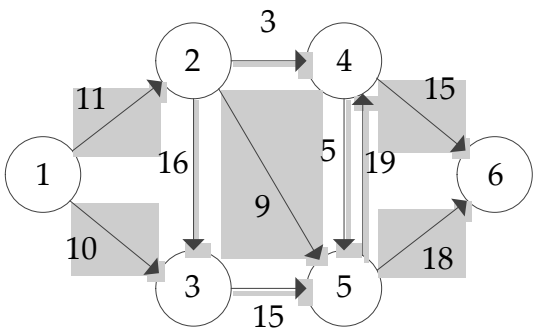
Exercício 3.8 * Caminho mais curto (algoritmo específico)

Indique qual o algoritmo adequado para resolver o problema árvore dos caminhos mais curtos com raiz em 1 e aplique-o.



Exercício 3.9 Variantes e extensões do problema de caminho mais curto (modelos + *solver*)

Considere a rede da figura.



a) Apresente um modelo de programação inteira para o problema de determinar o caminho entre 1 e 6 com o menor número de saltos.

b) Através do modelo de programação inteira e da utilização de *software*, obtenha os dois caminhos disjuntos nos arcos, com menor custo total, entre 1 e 6.

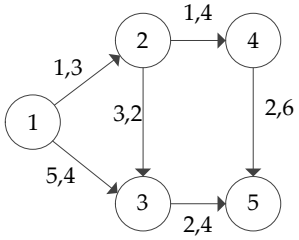
c) Apresente um modelo de programação inteira para o problema de determinar o caminho entre 1 e 6 em que o arco mais longo do caminho tem a menor distância possível. Por exemplo, o arco mais longo do caminho 1-2-4-6 tem distância 15, logo este caminho é melhor do que o caminho 1-3-5-6 cujo arco mais longo tem distância 18.

d) Através da utilização de *software* obtenha a solução óptima do problema da alínea anterior.

e) Através do modelo de programação inteira e da utilização de *software*, obtenha os 3 caminhos mais curtos entre 1 e 6.

Exercício 3.10 Variantes e extensões do problema de caminho mais curto (modelos + *solver*)

Considere a rede da figura em que junto aos arcos estão indicados o seu comprimento e a sua duração. Por exemplo, o arco (1,3) tem um comprimento de 1 unidade de distância e uma duração de 3 unidades de tempo.



a) Obtenha a árvore dos caminhos mais curtos (i.e., respeitante ao primeiro valor associado aos arcos) com raiz em 1 através do algoritmo que achar mais adequado, justificando a sua escolha.

b) Qual o algoritmo que utilizaria para determinar a árvore dos caminhos mais curtos se existisse o arco (3,2)? Justifique.

c) Apresente um modelo de programação linear para o problema de determinar o caminho mais rápido (i.e., respeitante ao segundo valor associado aos arcos) entre 1 e 5.

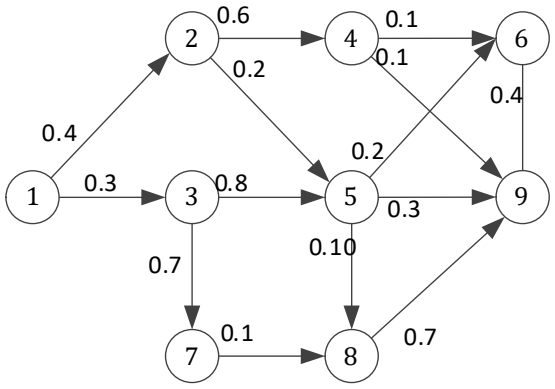
d) Quais as alterações necessárias ao modelo da alínea anterior para incorporar que se pretende o caminho mais rápido que não ultrapasse as 6 unidades de distância. Por inspecção, obtenha a solução óptima.

e) Apresente um modelo de programação inteira para o problema de determinar os dois caminhos disjuntos *nos arcos* entre 1 e 5 cuja soma dos comprimentos é menor. Por inspecção, obtenha a solução óptima.

e) Apresente um modelo de programação inteira para o problema de determinar os dois caminhos disjuntos *nos nodos* entre 1 e 5 cuja soma dos comprimentos é menor. Por inspecção, obtenha a solução óptima.

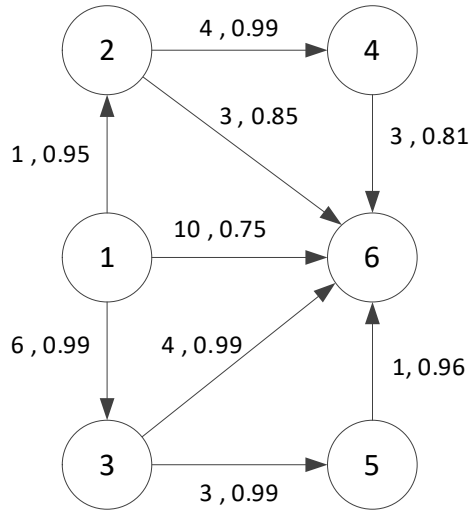
Exercício 3.11 Caminho mais fiável

Obtenha o caminho mais fiável entre 1 e 9 sabendo que a fiabilidade de um arco tem o valor indicado na rede.



Exercício 3.12 Caminho preferido bi-objectivo (duração e fiabilidade)

Na rede seguinte, junto a cada arco são dadas a sua duração (numa unidade de tempo) e a sua fiabilidade (probabilidade de o arco não falhar), por esta ordem. Considere o problema do caminho preferido bi-objectivo em que um objectivo é minimizar a duração do caminho e o outro objectivo é maximizar a fiabilidade.



a) Por inspecção, identifique todos os caminhos entre 1 e 6. Determina a duração e a fiabilidade de cada caminho.

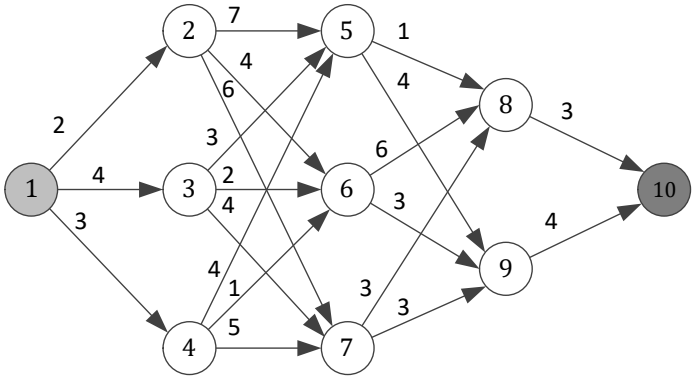
b) Indique os caminhos dominados e os caminhos eficientes.

c) Sabendo que, para o agente de decisão, 10 unidades de tempo são equivalentes a 0.1 de fiabilidade, indique a solução preferida.

d) Após normalizar os objectivos, indique o caminho que tem a menor distância à ideal.

Exercício 3.13 Caminho preferido bi-objectivo (distância e minmax)

Considere a rede seguinte e o problema do caminho preferido entre 1 e 10 com os objectivos de minimizar a distância e minimizar o comprimento do arco mais comprido (objectivo minmax).



a) Obtenha um conjunto de soluções eficientes por aplicação do método de geração através de pesos com 6 optimizações. Para valores máximos a usar na normalização, considere o valor do caminho mais longo e o arco com maior comprimento.

b) Obtenha a solução preferida de acordo com o método da distância ao ideal.

c) Obtenha uma solução óptima lexicograficamente considerando como objectivo mais importante o objectivo de minimizar o comprimento do arco mais comprido.

Exercício 3.14 Aplicação: Substituição de frota

Uma empresa pretende definir a sua política de substituição de uma frota de veículos para os próximos cinco anos tendo em conta o custo de aquisição da frota, o custo de manutenção e o valor residual. Em qualquer dos anos, o custo de renovação total da frota é de 15 U.M. O custo de manutenção da frota no primeiro ano é de 5 U.M. e aumenta em 2 U.M. em cada ano subsequente. O valor residual da frota é de 10 U.M. no final do primeiro ano, 7, 5 e 3 no final dos segundo, terceiro e quarto anos e de 2 no final do quinto ano. Este problema pode ser modelado como um problema de caminho mais curto em que ao início do ano *i* está associado um nodo e ao arco *ij* a substituição da frota nos anos *i* e *j*. Obtenha a política óptima de substituição.

Exercício 3.15 Aplicação: Lotes de produção

Uma fábrica pretende efectuar o seu planeamento de produção para as próximas seis semanas de um determinado produto, tendo em conta uma estimativa, para cada semana, da procura, do custo de produção (por unidade) e do custo de armazenamento (por unidade e semana). Esses valores são dados na tabela. Considere que as unidades produzidas durante uma semana estão disponíveis para satisfazer a procura dessa mesma semana (e eventualmente de semanas seguintes) e portanto não incorrem em custos de armazenamento.

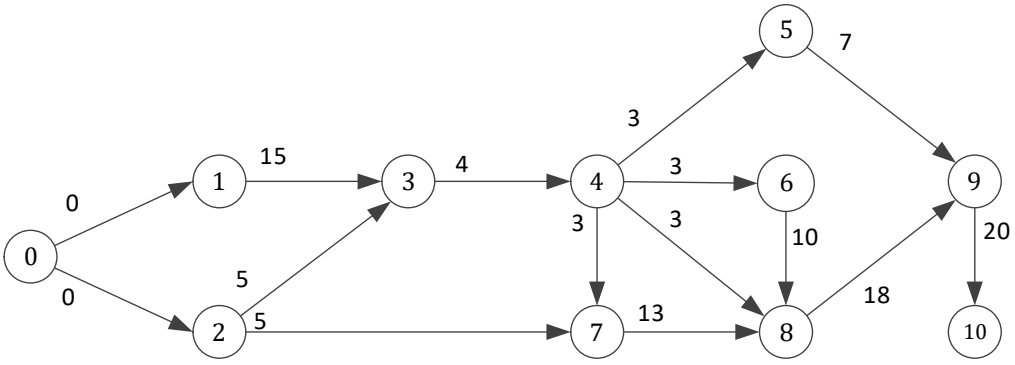
Modele este problema como um problema de caminho mais curto. Para tal, considere uma rede em que cada nodo está associado ao final de uma semana / início da seguinte. O arco ij corresponde a produzir na semana de i para satisfazer a procura de i até j-1. Por exemplo o arco 12 corresponde a produzir na semana 1 para satisfazer a procura da própria semana. O arco 13 corresponde a produzir em 1 para satisfazer a procura das semanas 1 e 2.

Semana	1	2	3	4	5	6
Procura	20	80	60	26	40	32
Custo de produção (U.M./unidade)	3	6	5	4	3	2
Custo de armazenamento (U.M./(unidade.semana))	9	1	4	2	2	–

Exercício 3.16 Aplicação: Escalonamento de projectos (1)

Considere o caso de uma empresa de construção civil que vai iniciar a construção de um edifício. A informação relativa às actividades, duração e relações de precedência são dadas na tabela representadas na figura.

Actividade	Descrição	Duração	Actividades imediatamente precedentes
1	Fundações	15	-
2	Medições	5	-
3	Placas	4	1,2
4	Estrutura	3	3
5	Telhado	7	4
6	Electricidade	10	4
7	Aquecimento e ar condicionado	13	2,4
8	Pintura	18	4,6,7
9	Acabamentos	20	5,8



O problema de escalonamento de projectos é um problema de caminho mais longo mas que pode ser transformado no problema de caminho mais curto por a rede ser acíclica. A transformação consiste em considerar os custos simétricos dos originais.

Obtenha a duração mínima do projecto.

Exercício 3.17 Aplicação: Escalonamento de projectos (2)

Considere um projecto constituído pelas actividades e relações de precedência dadas na Tabela seguinte. Obtenha a menor duração possível do projecto.

Actividade	Duração	Predecessoras imediatas
1	5	2
2	3	–
3	2	1
4	3	3
5	4	1
6	8	4
7	5	8
8	2	1
9	2	6,7,10
10	3	8,5

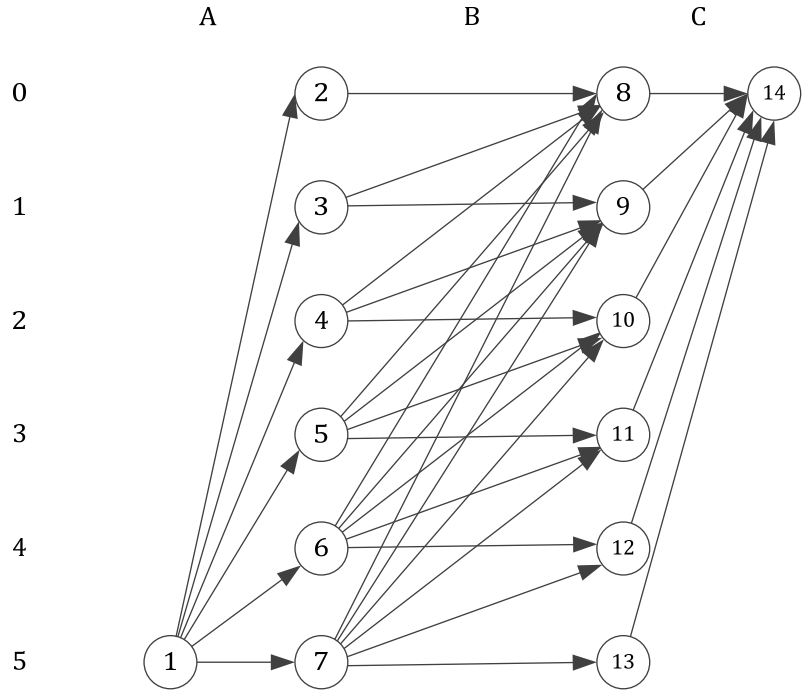
Exercício 3.18 Aplicação: Programação dinâmica

A programação dinâmica é uma abordagem a problemas de optimização em que estes são modelados como um conjunto de decisões sequenciais (cada uma associada a um estágio) que são condicionadas por um conjunto de variáveis (variáveis de estado). Se o objectivo puder ser representado pela soma dos valores das decisões de cada estágio, o problema de programação dinâmica é equivalente ao problema de caminho mais curto numa rede acíclica.

Considera-se o exemplo de uma organização de saúde que tem disponíveis cinco equipas médicas para alocar a três países com vista à melhoria dos cuidados de saúde neles prestados. A medida de desempenho utilizada é o produto do aumento da esperança de vida (em anos) pela população do país. Os dados relativos ao problema (em milhões) são dados na tabela.

Número de equipas médicas	País		
	A	B	C
0	0	0	0
1	45	20	50
2	70	45	70
3	90	75	80
4	105	110	100
5	120	150	130

Neste exemplo, as decisões sequenciais são quantas equipas enviar para o país A, o B e o C. O que condiciona o envio de equipas para um país é o número de equipas disponíveis. É essa a variável de estado. A representação em rede do problema é dada na figura e os custos associados aos arcos na tabela.



	2	3	4	5	6	7
1	120	105	90	70	45	0

	8	9	10	11	12	13
2	0	-	-	-	-	-
3	20	0	-	-	-	-
4	45	20	0	-	-	-
5	75	45	20	0	-	-
6	110	75	45	20	0	-
7	150	110	75	45	20	0

	14
8	0
9	50
10	70
11	80
12	100
13	130

3.8 Bibliografia

- R. Ahuja, T. Magnanti, J. Orlin, "Network Flows", 1993, Prentice-Hall.
- Masud, A. S., & Ravindran, A. R. (2008). Multiple criteria decision making. in Ravindran, A. R. (ed.) "Operations Research and Management Science Handbook", CRC press.
- Matos, M. Apontamentos da disciplina de Metodologias de Ajuda à Decisão, Mestrado em Engenharia Electrotécnica e de Computadores, Faculdade de Engenharia da Universidade do Porto, 2005/06, Disponíveis em https://paginas.fe.up.pt/~mam/MAD.html.

3.9 Resultados de aprendizagem

Representar problemas de caminho mais curto (inlcuindo variabtes e extensões) através de modelos de programação inteira.

Resolver problemas de caminho mais curto com software, nomeadamente o *Solver* e *openSolver* para *Excel* e o *IBM ILOG CPLEX Optimization Studio*.

Aplicar algoritmos específicos para resolver instâncias de pequena dimensão de problemas de caminho mais curto.

Identificar soluções dominadas e eficientes em problemas multi-atributo.

Aplicar métodos de optimização multi-objectivo para a obtenção de soluções eficientes em problemas multi-objectivo.