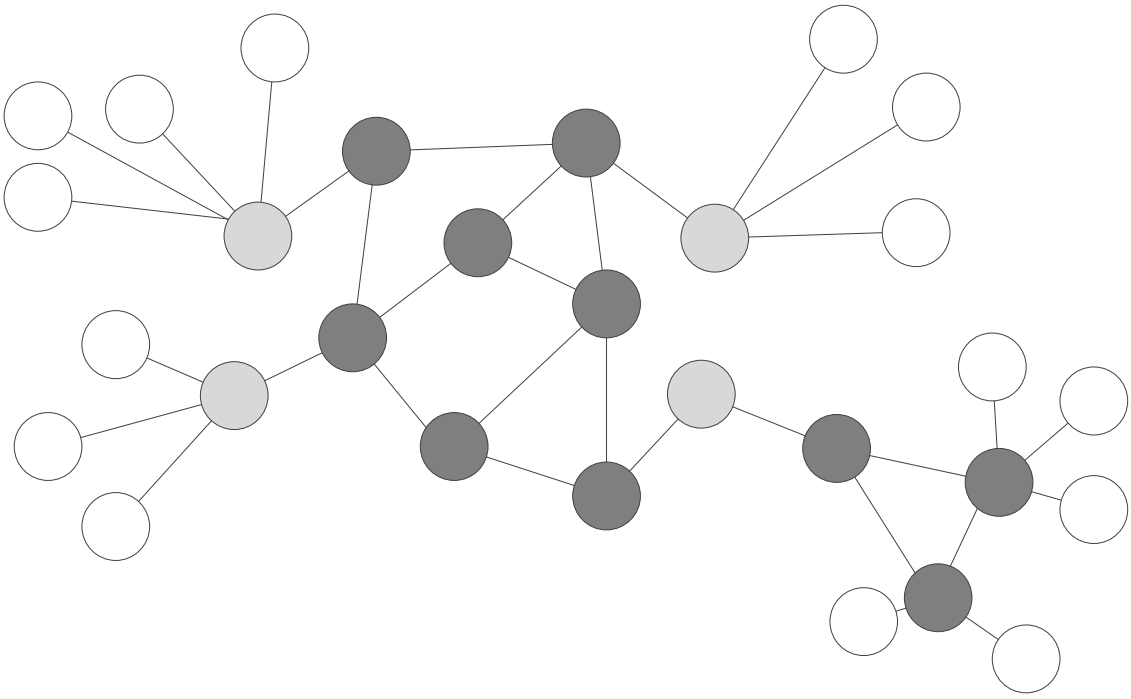
Universidade do Minho

Optimização de Redes



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 diz-se dominada quando existe uma outra solução que tem melhor ou igual valor em todos os objectivos, sendo melhor em, pelo menos, um objectivo. Uma solução é eficiente (ou óptima de Pareto) quando não é dominada. O conjunto das soluções eficientes designa-se por fronteira eficiente (ou fronteira de Pareto).

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.

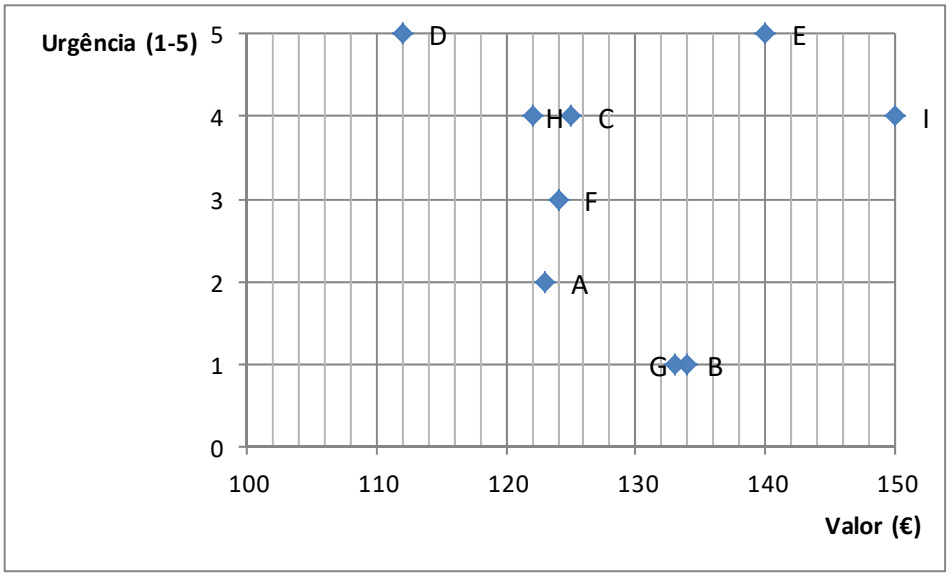


Figura 1.1. Representação no espaço dos objectivos do Exemplo 1.7.

■

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 prática 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

[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.

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 $cf(n)$.

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.

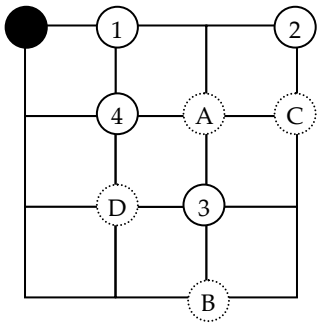
A empresa B vende cada vacina por 0.2€ mas não se responsabiliza pelo seu transporte para os Centros de Saúde. Esta empresa não assegura mais de 100 000 vacinas. Estima-se o custo de transporte por vacina das instalações da empresa B para cada um dos Centros de Saúde em 0.15€, 0.2€, 0.35€ e 0.4€ (pela 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 1.9 e representados no plano na Figura 1.3.

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

Tabela 1.9. Dados do exemplo da localização multi-atributo.

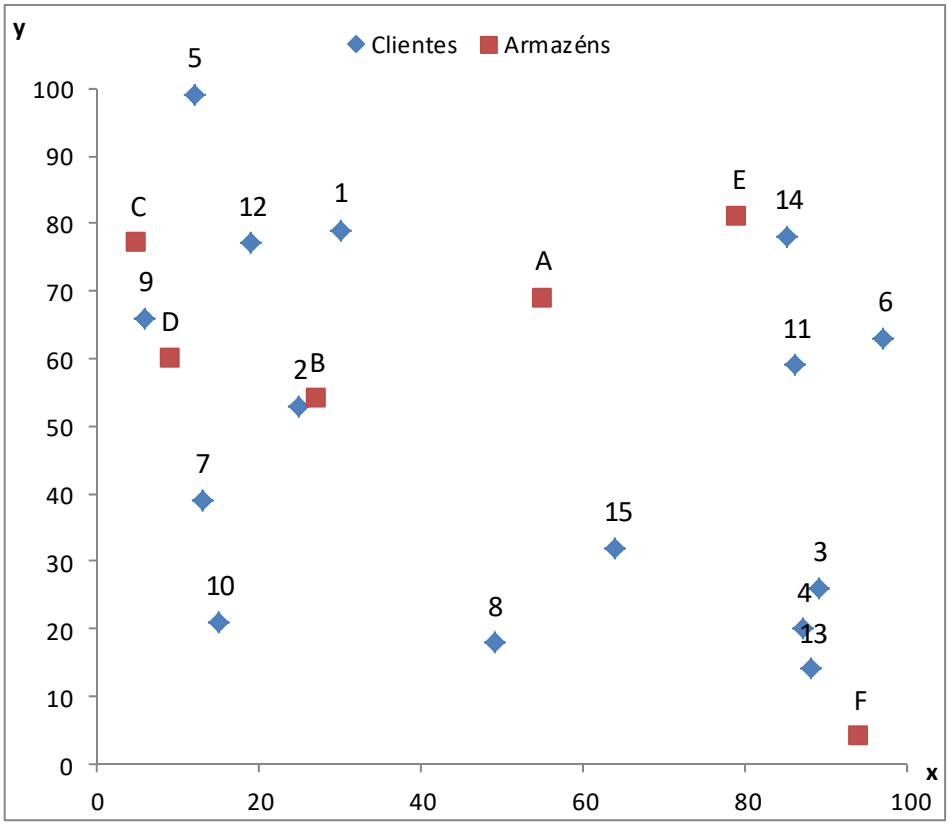


Figura 1.3. Representação no plano das localizações do exemplo da localização multi-atributo.

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 1.10.

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

Tabela 1.10. 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

1. Formular modelos de programação linear de actividades de recursos e transportes.
2. Formular modelos de programação inteira de mochila e afectação.
3. Aplicar uma heurística construtiva para o problema da mochila.
4. Identificar soluções dominadas e eficientes em problemas multi-atributo.
5. 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 $A = \{\{1,2\}, \{1,3\}, \{2,4\}, \{3,2\}, \{3,4\}, \{4,5\}, \{5,2\}, \{5,6\}, \{6,4\}, \{6,7\}, \{7,3\}\}$.

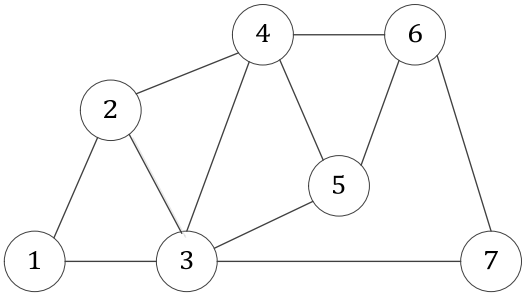


Figura 2.1. Um grafo com 7 vértices e 11 arestas.

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.

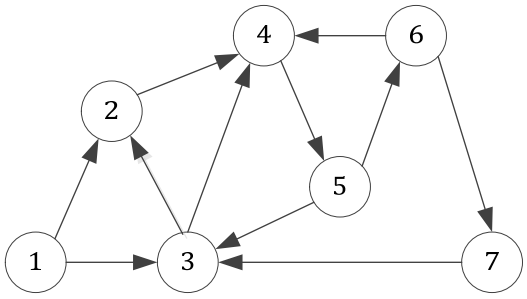
Um **caminho** (elementar ou simples) é uma sequência de vértices distintos e das arestas que os unem.

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.

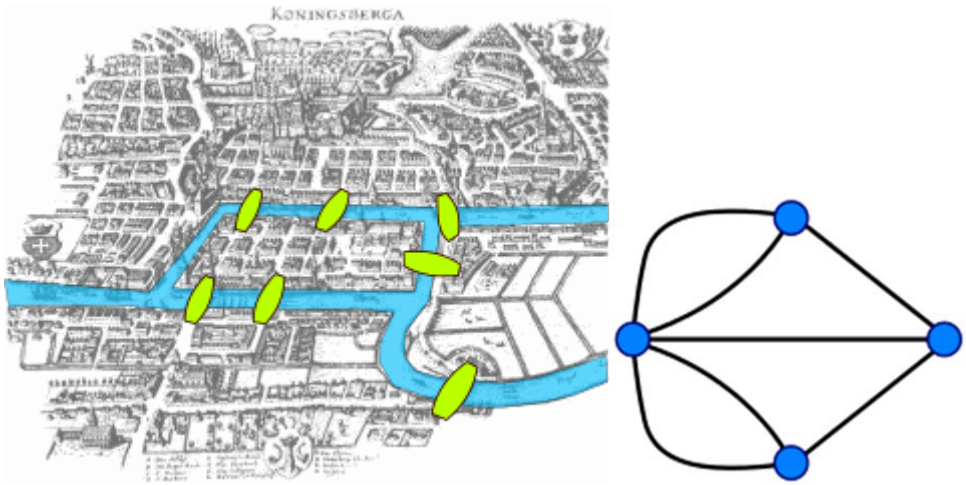


Figura 2.3. Ilustração da cidade de Könisberg e grafo do Problema das pontes Könisberg: ilustração da cidade e grafo correspondente.[2]

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.

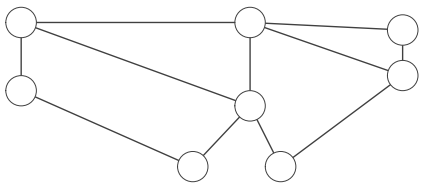


Figura 2.4. Grafo conexo.

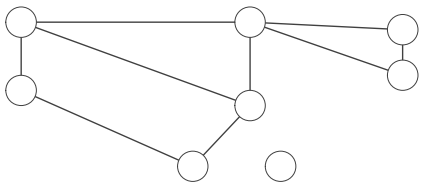


Figura 2.5. Grafo não conexo.

[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 $N_2 = \{2,3,6,7\}$.

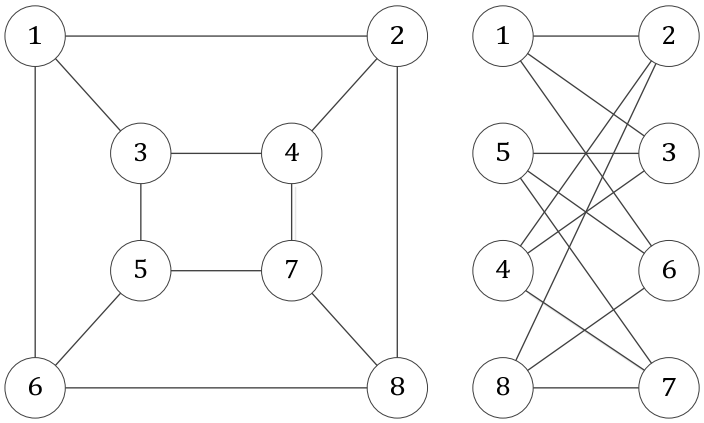


Figura 2.6. Um grafo bipartido representado de duas formas.

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

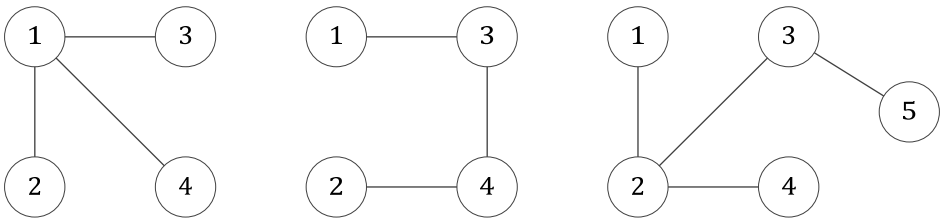


Figura 2.7. Três árvores.

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).

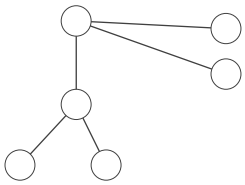


Figura 2.8. Um subgrafo do grafo da Figura 2.4 que é uma árvore mas não de suporte.

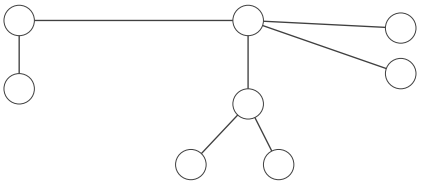


Figura 2.9. Um subgrafo do grafo da Figura 2.4 que é uma árvore de suporte.

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 $\{(1,3), (2,4), (3,2)\}$.

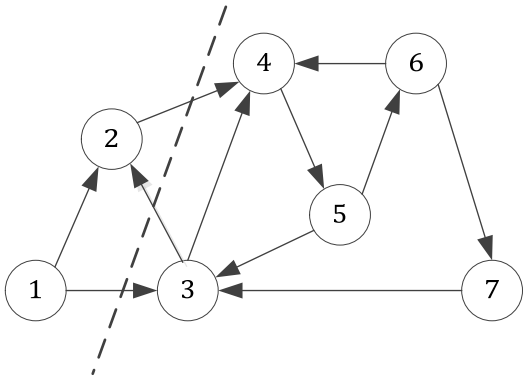


Figura 2.10. Um corte.

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.

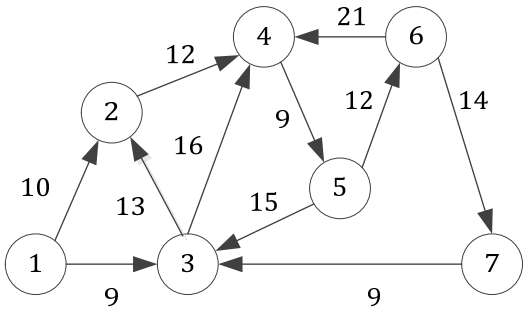


Figura 2.11. Rede do exemplo.

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.

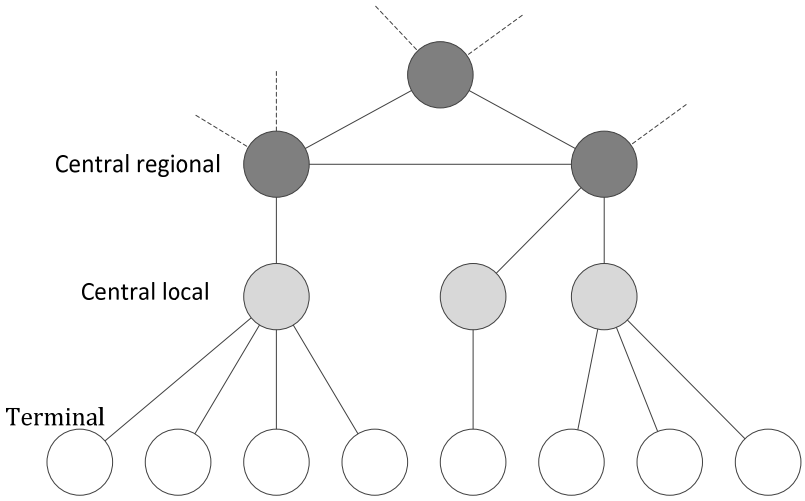


Figura 2.12. Rede telefónica genérica.

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.

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.

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.

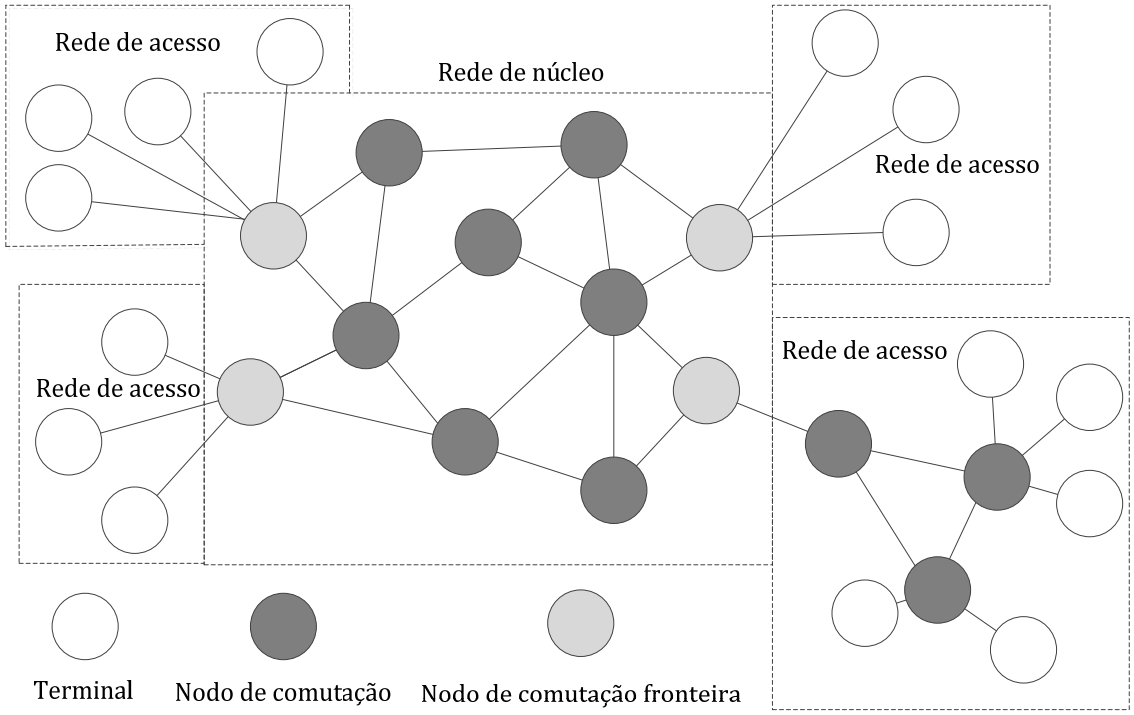


Figura 2.13. Rede de computadores genérica.

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.

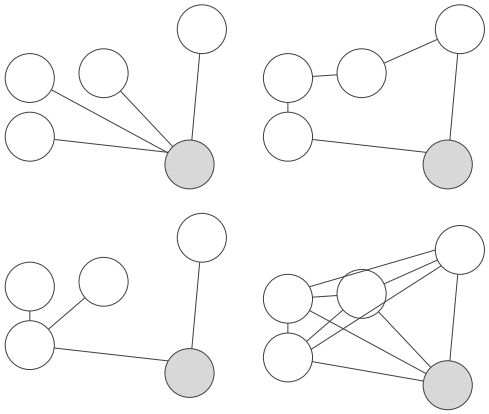


Figura 2.14. Exemplos de topologias de rede, partindo do canto superior esquerdo com a orientação dos ponteiros do relógio: estrela, anel, completa e árvore.

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.

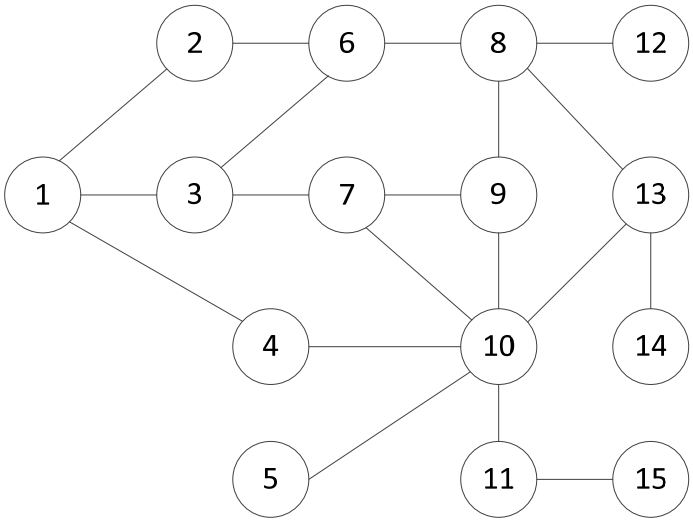


Figura 2.15. Exemplo de rede social.

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.

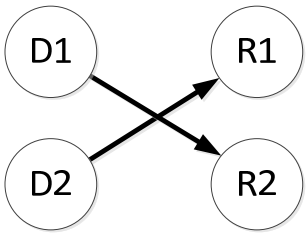


Figura 2.16. Transplantes cruzados.

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).

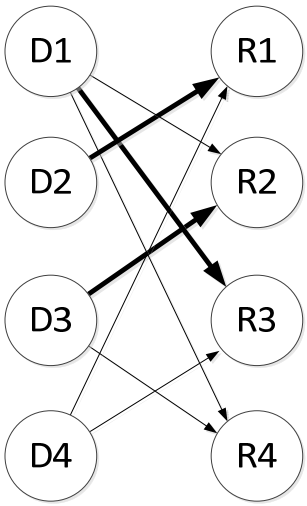


Figura 2.17. Problema com quatro pares de dadores representando numa rede em que cada nodo está associado a um dador ou receptor.

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.

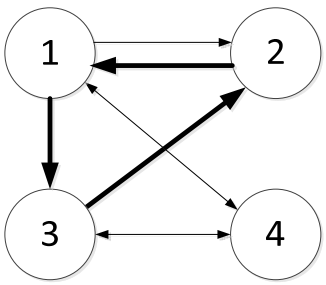


Figura 2.18. Problema com quatro pares de dadores representando numa rede em que cada nodo está associado a par dador-receptor.

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.

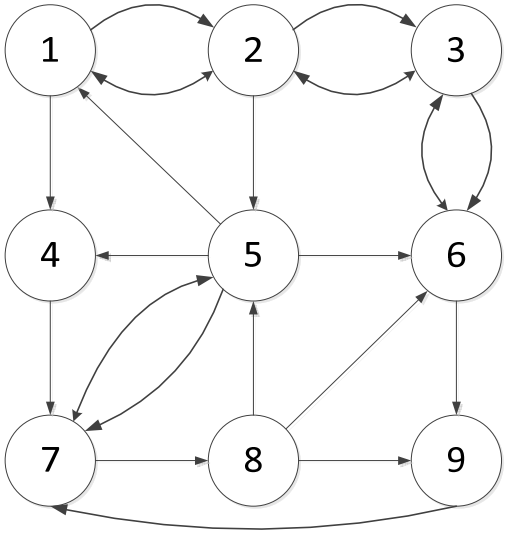


Figura 2.19. Instância do problema dos transplantes renais cruzados com 9 pares.

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.

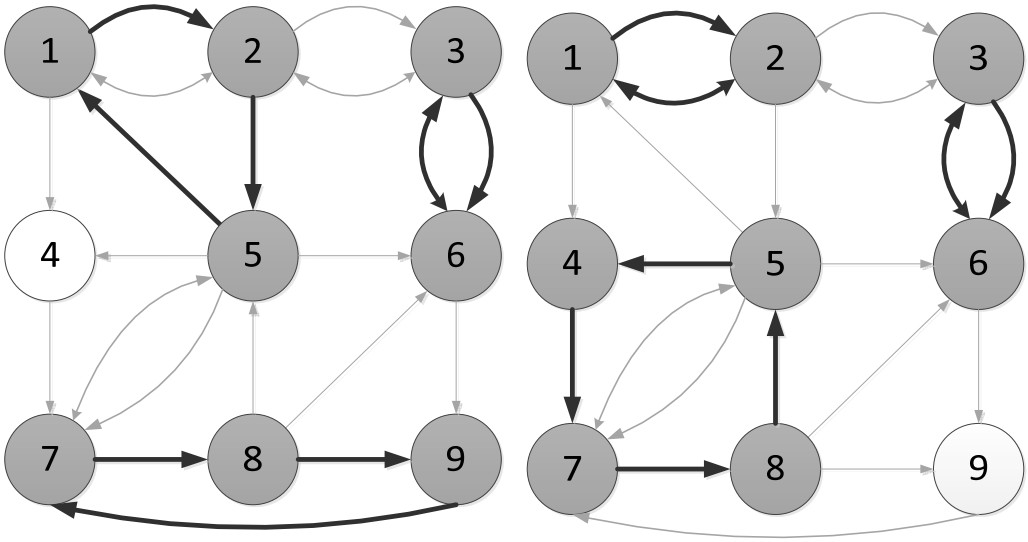


Figura 2.20. Dois conjuntos de transplantes cruzados possíveis.

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.

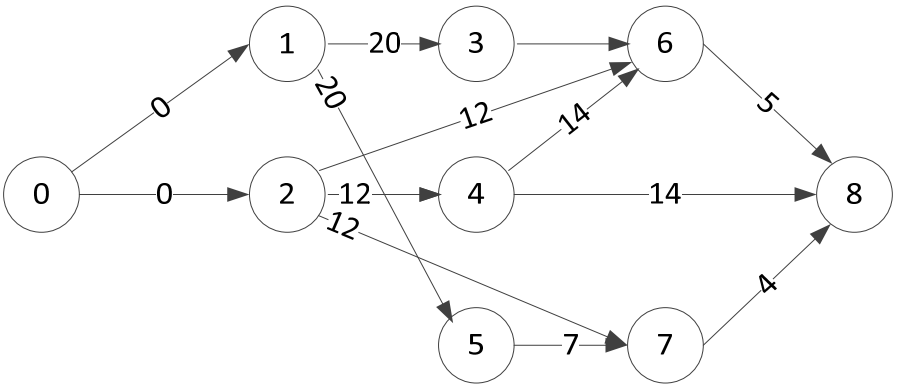


Figura 2.21. Exemplo de uma rede de escalonamento de projectos.

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.

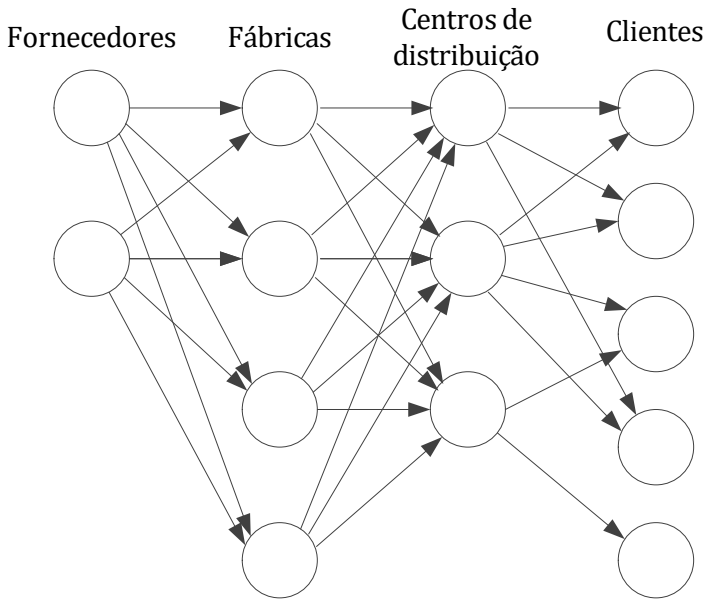
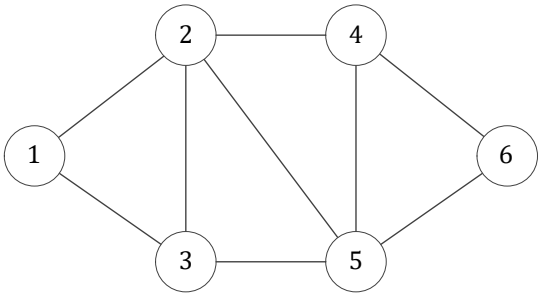


Figura 2.22. Exemplo de uma rede que representa uma cadeia de abastecimento.

2.5 Exercícios

Exercício 2.1 Definições - rede não orientada

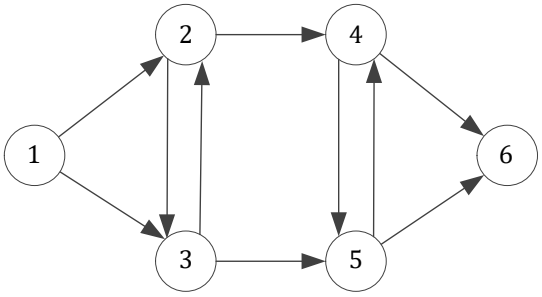
Considere o grafo da figura.



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 entre dois nodos

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 entre dois nodos 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.

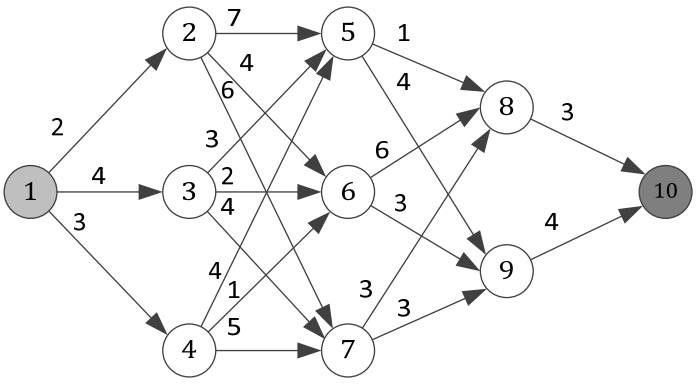


Figura 3.1. Instância do problema do caminho mais curto entre dois nodos (a cinzento claro o nodo inicial e a cinzento escuro o nodo final).

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 51), 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 27), 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.

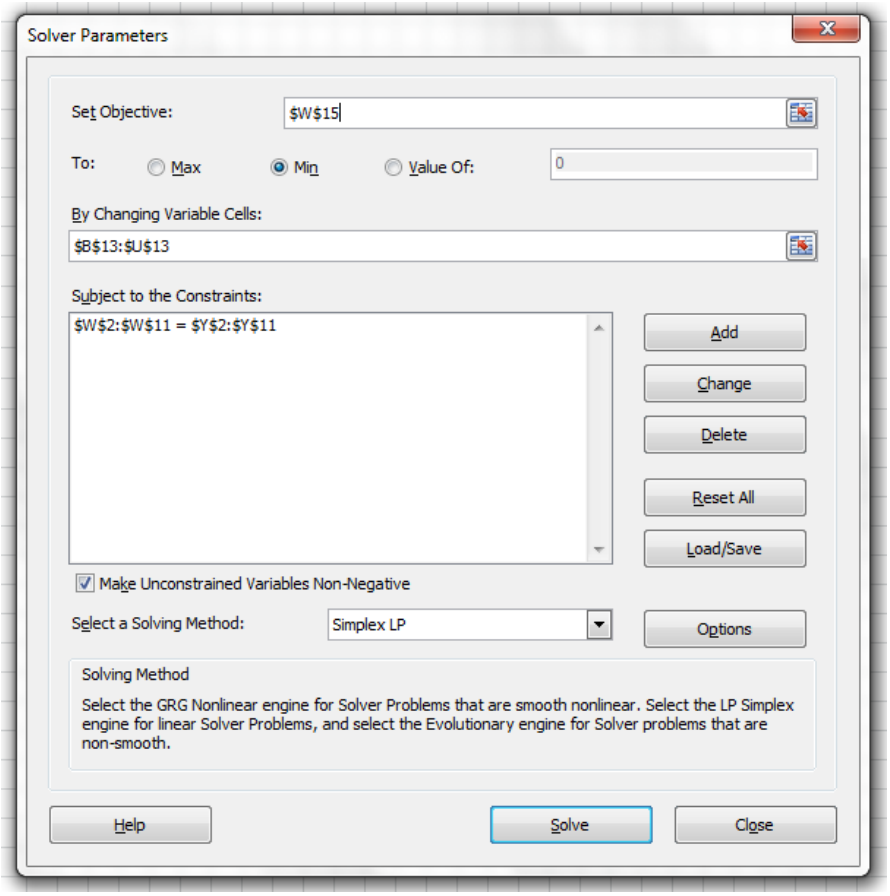
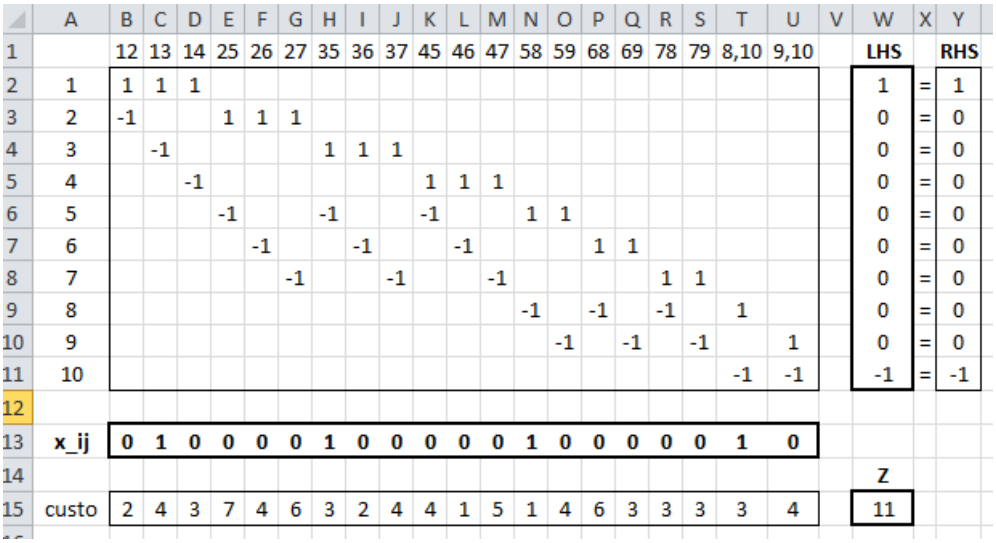


Figura 3.3. Parâmetros do Solver do Excel.

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.

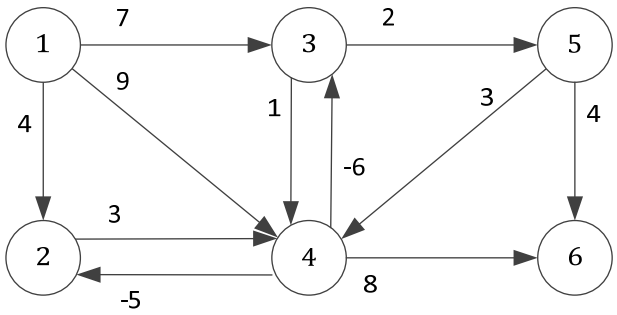


Figura 3.5. Exemplo de uma rede com ciclos negativos.

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).

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 $4 - 2 - 4$.

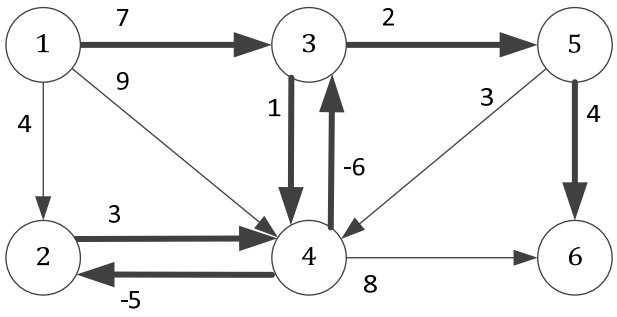


Figura 3.6. Solução óptima de um modelo de programação inteira (arcos do caminho mais curto não elementar a carregado) que não corresponde a um caminho elementar.

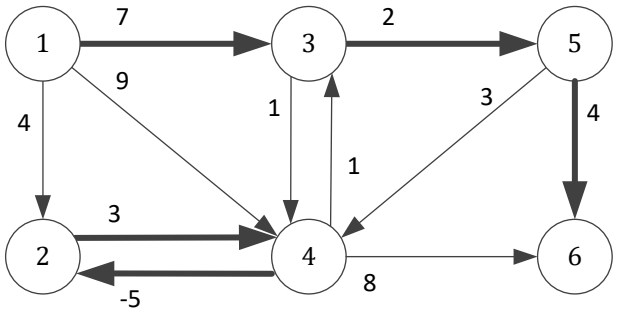


Figura 3.7. Solução óptima de um modelo de programação inteira (arcos do caminho mais curto não elementar a carregado) que não corresponde apenas a um caminho mas a a um caminho e um ciclo.

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].

3.3.1 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 23). 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.

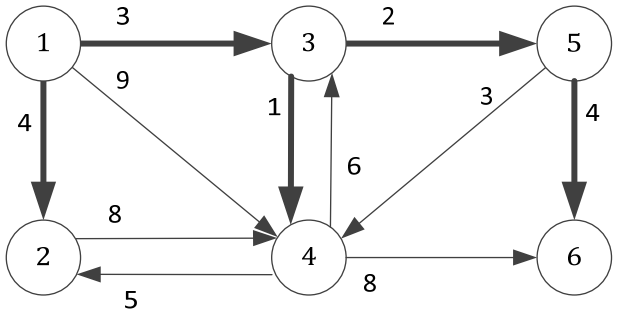


Figura 3.8. Uma árvore de caminhos mais curtos.

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.

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

[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.

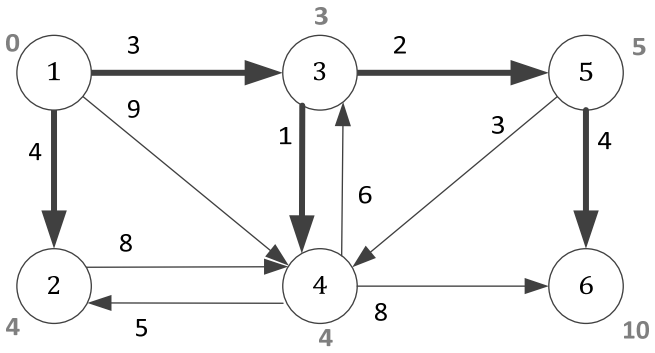


Figura 3.9. Comprimentos obtidos com base numa árvore.

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:

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

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:

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

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.

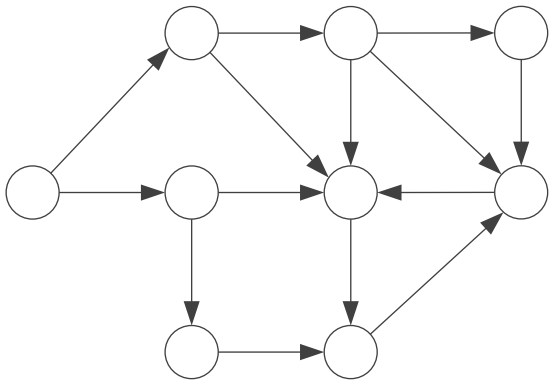


Figura 3.10. Exemplo de instância para ordenação topológica.

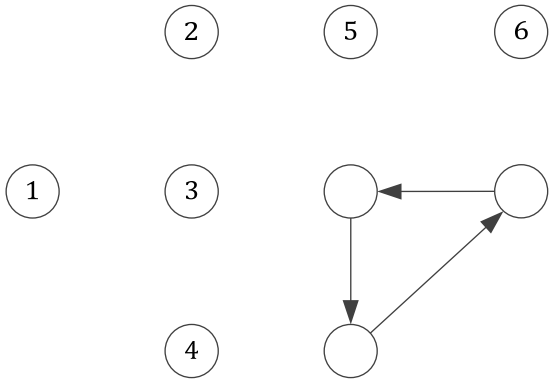


Figura 3.11. Resultado da aplicação do algoritmo de ordenação topológica.

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.

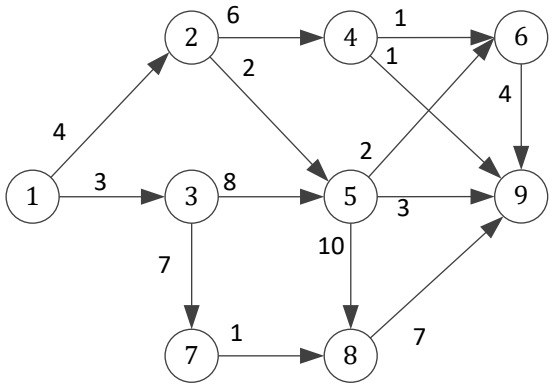


Figura 3.12. Instância de problema do caminho mais curto com rede já ordenada topologicamente.

		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.

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.

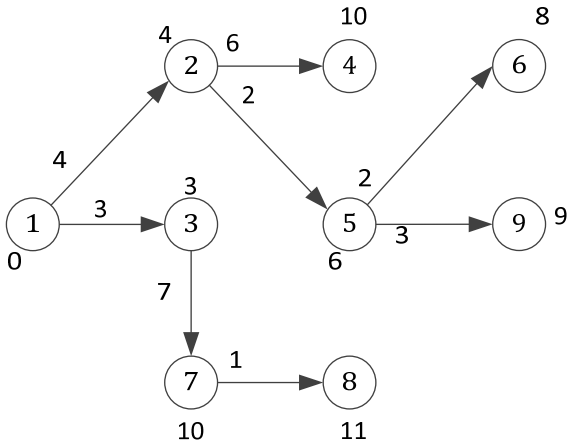


Figura 3.13. Árvore de caminhos mais curtos numa rede acíclica.

```
// 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
```

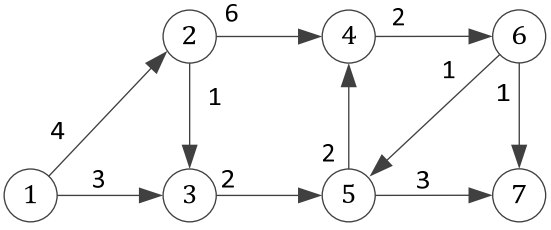


Figura 3.14. Instância para aplicação do algoritmo de Dijkstra.

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.

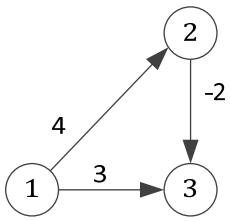


Figura 3.15. Exemplo de rede com custos negativos.

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
```

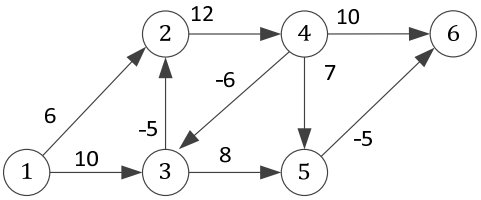


Figura 3.16. Exemplo de rede com custos negativos.

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.

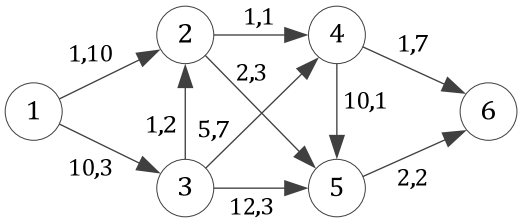


Figura 3.17. Rede de um problema de caminho mais curto com um recurso.

Um modelo de programação inteira geral é:

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

$$sujeito\ a:$$

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

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

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

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 $t_{ij} = 1$:

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

$$sujeito\ a:$$

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

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

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

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:

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

$$sujeito\ a:$$

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

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

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

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.

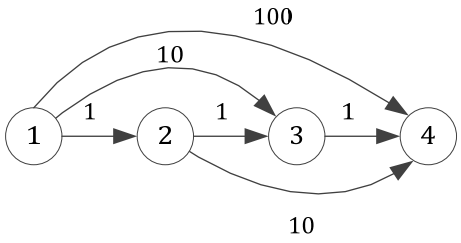


Figura 3.18. Ilustração da diferença do problema dos dois caminhos disjuntos com menor custo total e do problema dos dois caminhos mais curtos.

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.

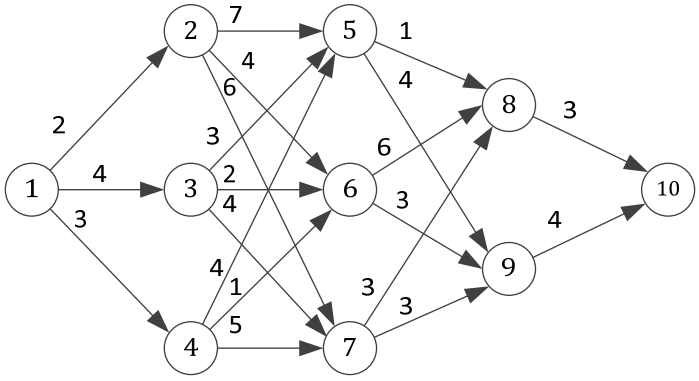


Figura 3.19. Rede do exemplo do caminho mais curto com objectivo minmax.

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 c não falhar é dada pelo produto da fiabilidade dos arcos que o compõem. Representando um caminho por c e o conjunto de caminhos por C, pretende-se resolver o problema

$$\max_{c\in C} \prod_{ij\in c} p_{ij} x_{ij}$$

Esta expressão é não linear e envolve a enumeração dos caminhos[5]. A sua utilização directa, em problemas de alguma dimensão, não é viável.

No entanto, é possível efectuar uma transformação para um problema equivalente, tendo em conta que um caminho que maximize a fiabilidade também maximiza o logaritmo da fiabilidade

$$\max_{c \in C} log\left(\prod_{ij \in c} p_{ij} x_{ij}\right)$$

Convertendo num problema de minimização

$$\min_{c \in C} -log\left(\prod_{ij \in c} p_{ij} x_{ij}\right)$$

e utilizando a propriedade do logaritmo de produtos ser a soma dos logaritmos

$$\min_{c \in C} \sum_{ij \in c} -(log p_{ij}) x_{ij}$$

Assim, o problema de determinar o caminho mais fiável pode ser modelado através de programação inteira

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

$$sujeito\ a:$$

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

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

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

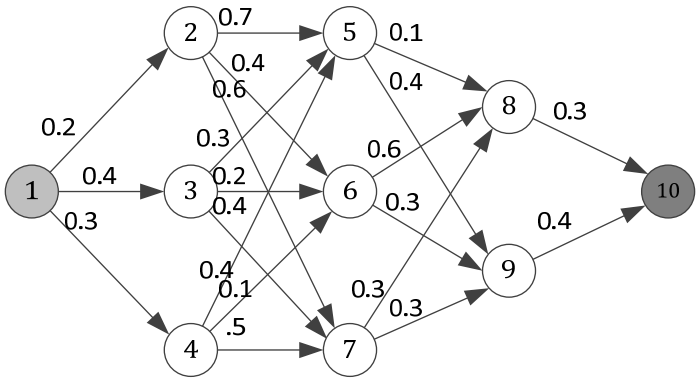


Figura 3.20. Instância do problema do caminho mais fiável (a cinzento claro o nodo inicial e a cinzento escuro o nodo final).

[5] Não seria necessário enumerar caminhos se se considerasse o modelo $Max\ z = \prod_{ij \in A} \left((1 - x_{ij}) + p_{ij} x_{ij}\right), sujeito\ a: x\ é\ um\ caminho$ mas este continuaria a ser não linear.

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

$$Min\ z_1 = \sum_{ij \in A} c_{ij} x_{ij}$$
$$Min\ z_2 = \sum_{ij \in A} t_{ij} x_{ij}$$
$$sujeito\ a:$$
$$\sum_{j:ij \in A} x_{ij} - \sum_{j:ji \in A} x_{ji} = \begin{cases} 1, & se\ i = o \\ 0, & se\ i \neq d, i \neq d \\ -1, & se\ i = d \end{cases}, \forall i \in N$$
$$x_{ij} \in \{0,1\}, \forall ij \in A$$

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 €).

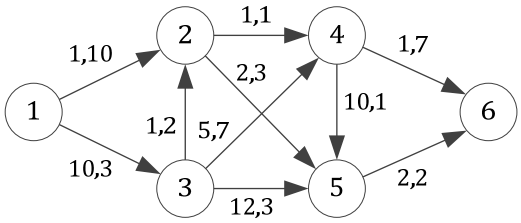


Figura 3.21. Rede do exemplo do caminho preferido b-objectivo.

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 (uma 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.

	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.

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

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;

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

$$sujeito\ a:$$

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

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

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:

$$Min\ s_1 + s_2$$
$$sujeito\ a:$$
$$x_{12} + 10\,x_{13} + x_{24} + 2\,x_{25} + x_{32} + 5\,x_{34} + 12\,x_{35} + 10\,x_{45} + x_{46} + 2\,x_{56} - s_1 = 3$$
$$10\,x_{12} + 3\,x_{13} + x_{24} + 3\,x_{25} + 2\,x_{32} + 7\,x_{34} + 3\,x_{35} + x_{45} + 7\,x_{46} + 2\,x_{56} - s_2 = 8$$
$$\sum_{j:ij\in A} x_{ij} - \sum_{j:ji\in A} x_{ji} = \begin{cases} 1, & se\ i = o \\ 0, & se\ i \neq d, i \neq d\ , \forall i \in N \\ -1, & se\ i = d \end{cases}$$
$$x_{ij} \in \{0,1\}, \forall ij \in A$$
$$s_1, s_2 \geq 0$$

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

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

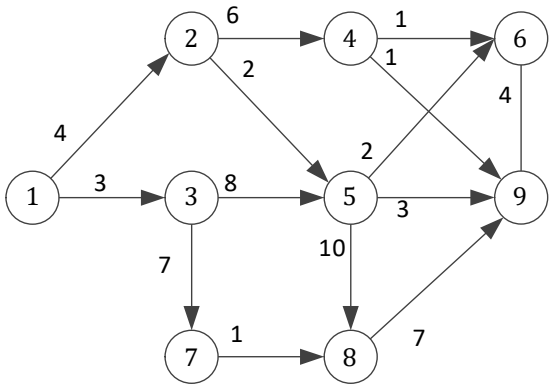
No exemplo dado na Figura 3.17., suponha-se que se pretende minimizar hierarquicamente o número de saltos e depois o custo.

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)

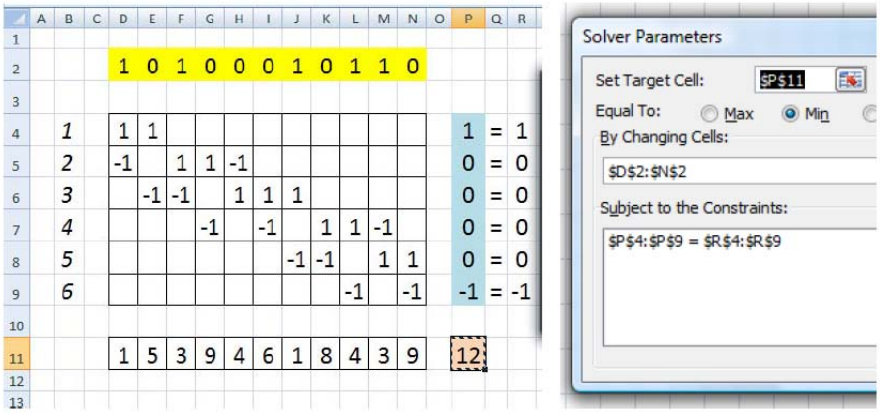
Indique o número de variáveis de decisão, o número de restrições e o número de coeficientes não nulos das variáveis de decisão nas restrições de um modelo de programação linear para o problema do caminho mais curto entre dois nodos definido numa rede com *n* nodos e *m* arcos.

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.

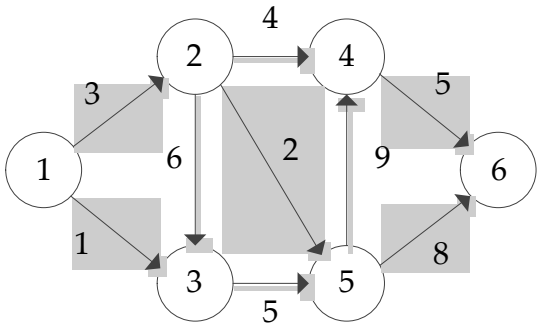
a) Represente a rede correspondente e a solução indicada na folha de cálculo.

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.

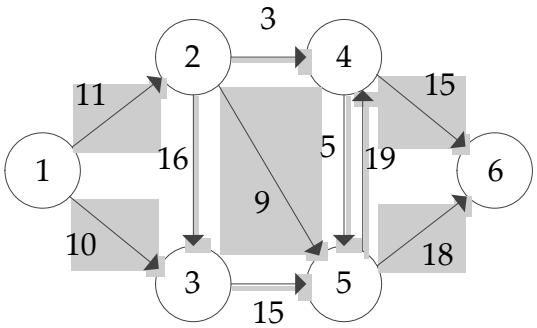


a) Apresente o modelo de programação linear para o problema do caminho mais curto entre 1 e 6.

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.

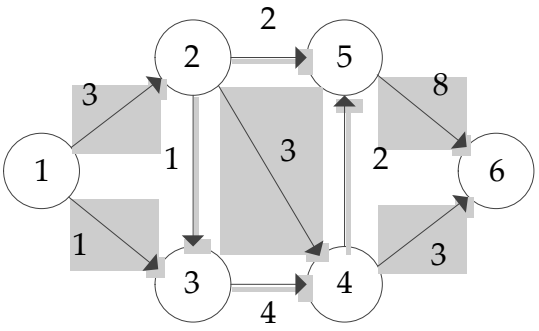
b) Obtenha a solução óptima de ambos os modelos com utilização de software.

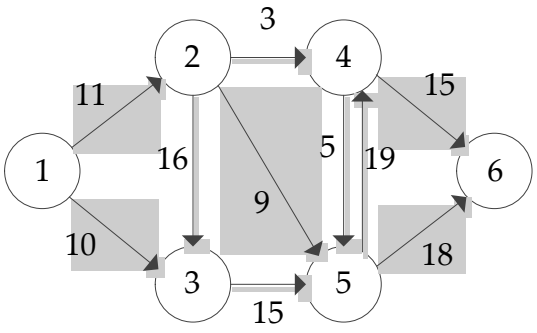
Exercício 3.6 Árvore dos caminhos mais curtos (algoritmos específicos)

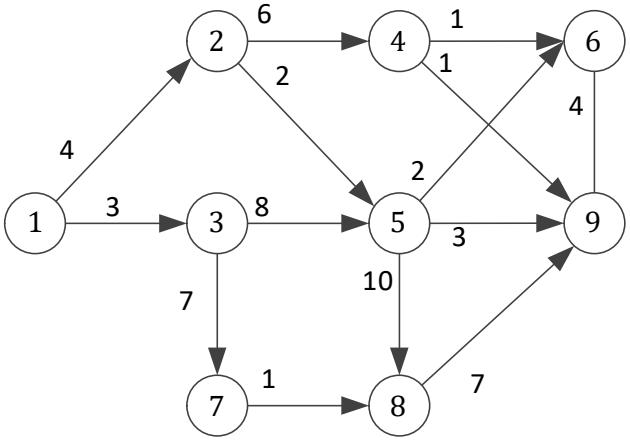
Para cada uma das quatro redes seguintes.

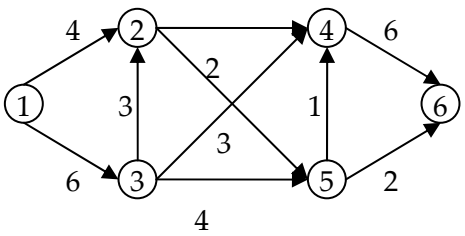
a) Mostre, através da aplicação de um algoritmo, que a rede é cíclica ou acíclica.

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



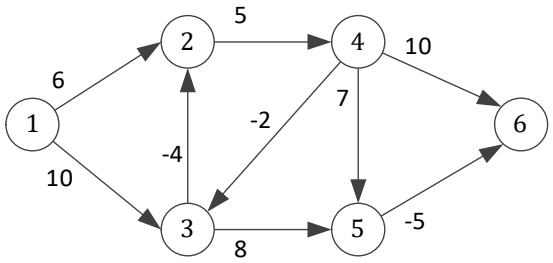






Exercício 3.7 * Caminho mais curto (algoritmo específico + modelo + *solver*)

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.

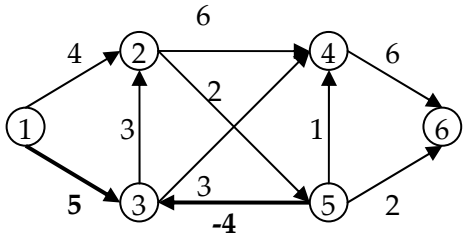
b) Indique a solução retornada pelo modelo de programação inteira discutido neste capítulo. A solução é admissível?

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.

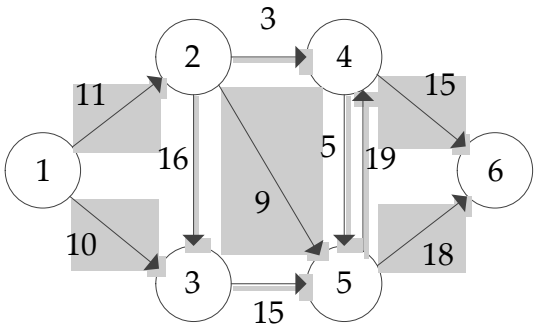
b) Através do modelo de programação inteira e da utilização de *software*, obtenha os dois caminhos disjuntos nos arcos, com menor custo total, entre 1 e 6.

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.

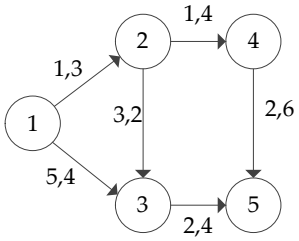
d) Através da utilização de *software* obtenha a solução óptima do problema da alínea anterior.

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.

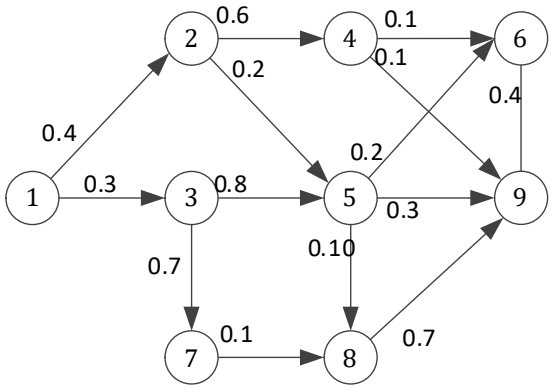
d) Quais as alterações necessárias ao modelo da alínea anterior para incorporar que se pretende o caminho mais rápido que não ultrapasse as 6 unidades de distância. Por inspecção, obtenha a solução óptima.

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.

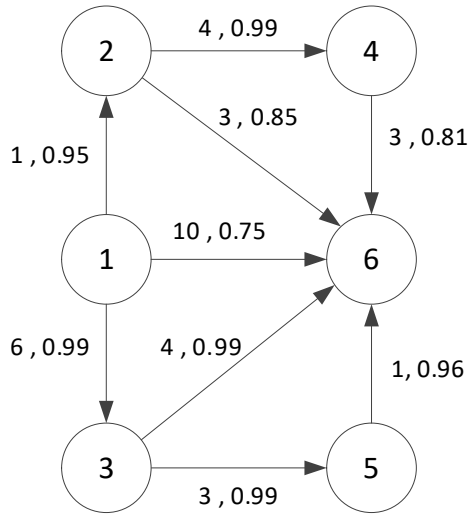
Exercício 3.11 Caminho mais fiável

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



Exercício 3.12 Caminho com objectivos comprimento 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.

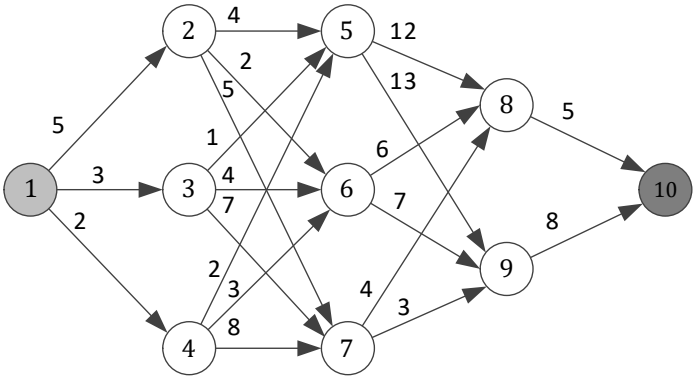
b) Indique os caminhos dominados e os caminhos eficientes.

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 com objectivos custo 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 4 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 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 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.

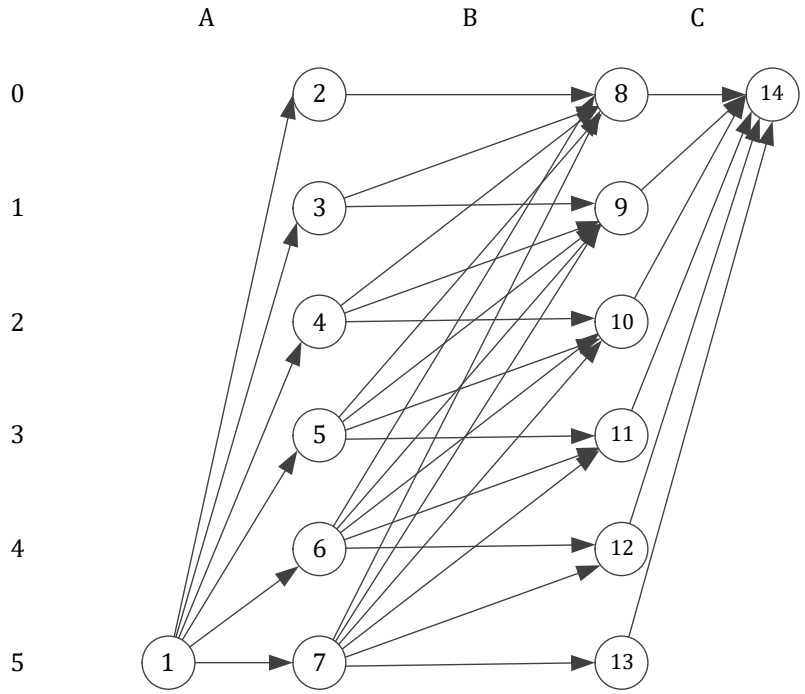
Número de equipas médicas	País		
	A	B	C
0	0	0	0
1	45	20	50
2	70	45	70
3	90	75	80
4	105	110	100
5	120	150	130

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



Exercício 3.16 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.

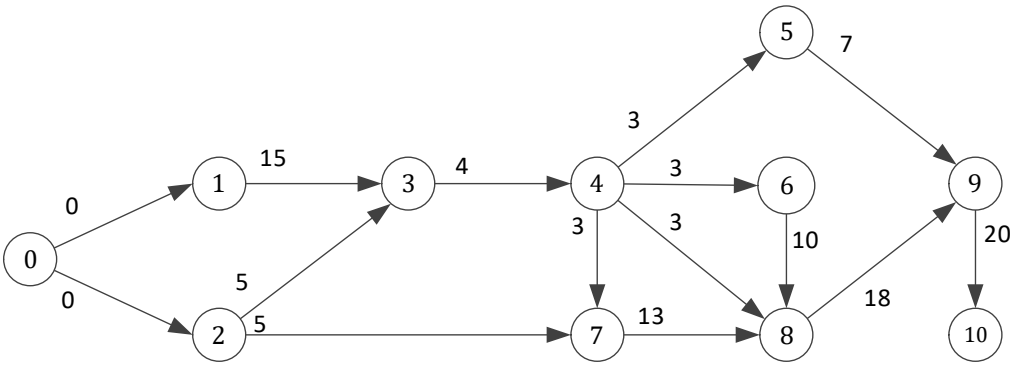
Modele este problema como um problema de caminho mais curto. Para tal, considere uma rede em que cada nodo está associado ao final de uma semana / início da seguinte. O arco ij corresponde a produzir na semana de i para satisfazer a procura de i até j-1. Por exemplo o arco 12 corresponde a produzir na 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 3.17 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.

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 3.18 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

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.
- Pióro, M., & Medhi, D. (2004). Routing, flow, and capacity design in communication and computer networks. Elsevier.

3.9 Resultados de aprendizagem

- Representar problemas de caminho mais curto (incluindo variantes 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.

4
Caixeiro viajante

4.1 Introdução

4.1.1 Definição

O problema do caixeiro viajante (PCV, ou *travelling salesman problem* - TSP) consiste em, dada uma rede com distâncias (custos) associadas às arestas, determinar um circuito elementar com menor distância (custo) total que inclua todos os vértices uma e uma só vez.

Dependendo da estrutura de custos, uma instância do PCV pode ser simétrica ou assimétrica. A instância da Figura 4.1 é simétrica porque a distância de um arco é independente do sentido em que este é percorrido (logo a instância pode ser definida numa rede não orientada). Instâncias em que a distância de um arco é diferente da distância do arco que liga os mesmos nodos mas em sentido inverso, dizem-se assimétricas (logo serão necessariamente representadas através de redes orientadas).

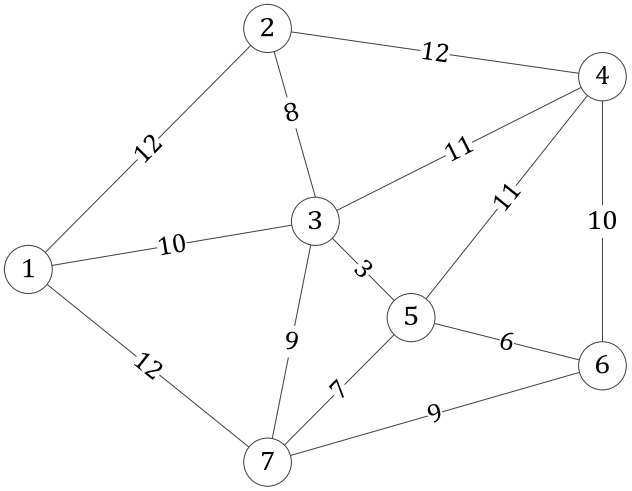


Figura 4.1. Instância do PCV representada por uma rede.

Uma instância do PCV pode também ser caracterizada através de uma matriz de distâncias como a representada na Tabela 4.1. Apenas se considera a matriz triangular superior (excluindo a diagonal) por a instância ser simétrica.

c_{ij}	1	2	3	4	5
1	-	6	3	13	11
2		-	7	10	9
3			-	8	15
4				-	4
5					-

Tabela 4.1. Instância do PCV simétrica representada por uma matriz.

Tipicamente, assume-se que existe um arco entre qualquer par de nodos. Este pressuposto não é restritivo, já que, caso não exista um arco entre dois nodos pode incluir-se um arco artificial com uma distância suficientemente elevada. Por exemplo, na instância da Figura 4.1 seriam incluídos os arcos artificiais 14, 15, 16, 25, 26, 27, 36 e 47, cada um com uma distância suficientemente elevada. A distância de um arco artificial pode ser calculada através do produto do número de vértices pelo maior custo de um arco mais um. No exemplo, 7*12+1=85. Dessa forma, nenhuma solução que inclua arcos artificiais é melhor do que uma solução sem nenhum arco artificial.

Uma solução que inclua arcos artificiais só poderá ser óptima se não existir nenhuma solução admissível sem arcos artificiais (caso contrário seria essa a solução óptima porque a distância de um arco artificial é maior do que a distância de qualquer solução admissível). Assim, se uma solução óptima inclui um arco artificial, o problema original (sem arcos artificiais) é impossível.

Note-se que esta distância artificial deverá ser o mais baixa possível, mas o seu cálculo não deverá ser demasiado elaborado. Uma alternativa ao cálculo agora descrito é usar uma distância artificial duas ordens de grandeza acima da maior distância. No exemplo, a distância artificial seria 1000.

Formalmente, o PCV define-se pelos seguintes parâmetros:

- $G = (N, A)$ grafo orientado (instância simétrica) ou não orientado (instância assimétrica)
- n número de vértices
- c_{ij} distância em que se incorre ao atravessar o arco ij, $\forall ij \in A$
- $c_{ij} \geq 0, \forall ij \in A$

No PCV simétrico: $c_{ij} = c_{ji}$, no PCV assimétrico: $\exists ij \in A: c_{ij} \neq c_{ji}$.

4.1.2 *PCV relaxado

A definição do PCV inclui a restrição de cada vértice ter de ser visitado uma e uma só vez. Considere-se agora o problema que se obtém ao permitir que um vértice seja visitado mais do que uma vez. Designa-se esse problema por PCV *relaxado*.

No caso de se verificar a desigualdade triangular, $c_{ij} \leq c_{ik} + c_{kj}, \forall i, k, j$, numa solução óptima do PCV relaxado um vértice nunca é visitada mais de uma vez. É esse o

caso em que os vértices correspondem a localizações e as distâncias são as distâncias euclideanas.

No caso de não se verificar a desigualdade triangular, pode haver soluções óptimas do PCV relaxado com menor valor do que soluções óptimas PCV. A título exemplificativo, na instância da Tabela 4.2, a solução óptima do PCV (assimétrico) é 1-2-3-1 com custo 22 e a solução óptima do PCV relaxado é 1-2-1-3-1 com custo 4. Note-se que a distância do arco 2-3 é maior do que a soma das distâncias entre 2-1 e 1-3 portanto a desigualdade triangular não se verificar (o caminho mais curto entre 2 e 3 é passar por 1 e não o directo).

	1	2	3
1	-	1	1
2	1	-	20
3	1	-	-

Tabela 4.2. Instância em que o valor óptimo do PCV é maior que o do PCV relaxado.

4.2 Motivação

A relevância do PCV pode ser justificada de diversas formas: i) pela sua importância histórica[6], ii) por ser um problema fundamental (no sentido de desenvolvimentos na sua resolução terem implicações em muitos outros problemas), iii) por ser um problema que tem servido de teste a um elevado número de técnicas de modelação e métodos de optimização, iv) por aparecer como um subproblema em muitos outros problemas e ainda v) pelas aplicações que encontra em problemas reais como, por exemplo, na definição de percursos para a recolha ou entrega de bens em diferentes localizações (e.g. correio, encomendas, notas para reabastecer ATM, armazéns automáticos), no fabrico de circuitos VLSI, no escalonamento de produção e no desenho de redes de computadores[7].

Uma possível abordagem para resolver o PCV é enumerar todas as soluções possíveis, calculando para cada uma o seu valor (distância total) escolhendo a solução com menor valor que, por definição, é a solução óptima. Dado que cada solução corresponde a uma permutação das cidades, o número de soluções possíveis é $n!$ para o

[6] Ver http://www.math.uwaterloo.ca/tsp/history/index.html para uma história do PCV.

[7] Ver http://www.math.uwaterloo.ca/tsp/apps/index.html para detalhes e exemplos adicionais. Em http://www.math.uwaterloo.ca/tsp/pubs/index.html, é apresentada a solução óptima para uma instância em que a cada um de 24727 vértices corresponde um pub do Reino Unido.

PCV assimétrico e $n!/2$ para o PCV simétrico (n é o número de vértices). Esta abordagem é irrealista, mesmo para instâncias pequenas – dezenas de cidades, como se constata pelos valores apresentados na Tabela 4.3.

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 4.3. Número de soluções para instâncias do PCV com diferentes dimensões.

No entanto, são resolvidas instâncias de muito maior dimensão como as representadas nas Figura 4.2 e Figura 4.3 (ambas com origem em problemas de determinar a sequência pela qual uma máquina deve perfurar um conjunto de pontos numa placa - relevantes no fabrico de circuitos impressos). A instância da Figura 4.3 é a maior instância resolvida até ao momento[8].

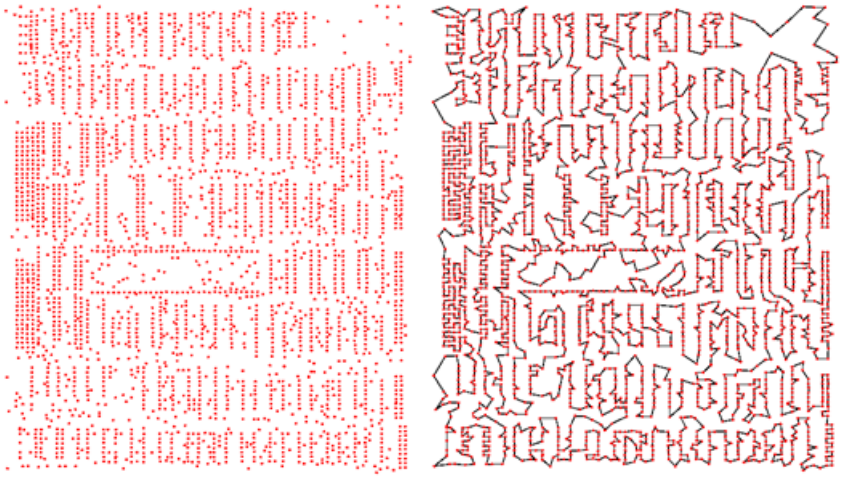


Figura 4.2. Instância e solução óptima de um PCV com 3038 vértices.

[8] http://www.math.uwaterloo.ca/tsp/index.html (de também onde provêm as imagens).

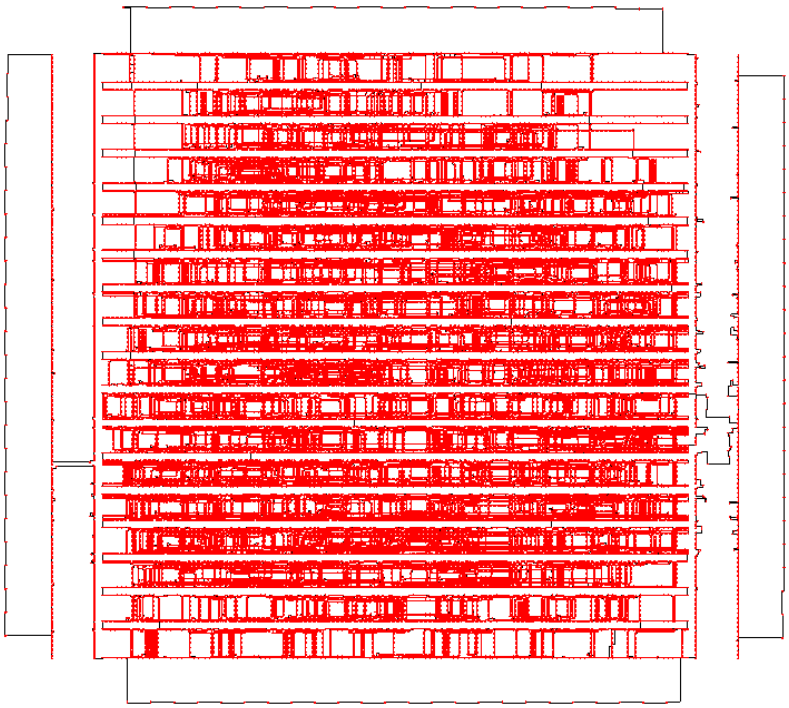


Figura 4.3. Solução óptima de um PCV com 85900 vértices.

Na base da resolução de problemas desta enorme dimensão estão métodos avançados baseados em programação inteira. Em concreto, na combinação do método de partição (*branch-and-bound*) com cortes e heurísticas.

Existe um prémio de 1000 dólares para quem resolver uma instância do PCV com 100 000 vértices derivada da imagem da "Mona Lisa"[9] (Figura 4.4).

Figura 4.4. Instância "Mona Lisa".

[9] http://www.math.uwaterloo.ca/tsp/data/ml/monalisa.html

4.3 Introdução a heurísticas

Em optimização, uma heurística[10] é um procedimento que se aplica a um problema específico com vista à obtenção (num tempo computacional aceitável) de uma solução de qualidade mas sem nenhuma garantia teórica de tal ser conseguido.

As heurísticas mais relevantes podem ser classificadas em três grupos: heurísticas de construção, heurísticas de pesquisa local e meta-heurísticas. A estes grupos correspondem diferentes níveis de exigências em termos de concepção e implementação (as construtivas são menos exigentes e as meta-heurísticas mais), de rapidez de execução (construtivas mais rápidas, meta-heurísticas menos) e de qualidade das soluções obtidas (construtivas pior, meta-heurísticas melhor).

Para um mesmo problema, podem existir várias heurísticas. O que as diferencia são os princípios gerais que as enquadram (i.e. o seu tipo) e a forma como exploram as características específicas do problema. Dada a elevada liberdade existente na concepção de uma heurística, o papel de quem a concebe é sempre relevante.

As heurísticas estão entre os métodos mais adequados, nalguns casos os únicos, para abordar problemas difíceis, de grandes dimensões, não lineares ou para os quais a obtenção de uma solução num tempo computacional reduzido é crucial.

4.4 Heurísticas construtivas

Uma heurística construtiva parte de uma solução vazia (ou parcial) e adiciona-lhe elementos, um a um, tendo em conta o objectivo do problema e a admissibilidade da solução a obter.

Para além da representação da solução (o que desde logo define os elementos que são considerados), o essencial de uma heurística construtiva reside na forma como é escolhido o elemento que é (tentativamente) adicionado à solução em cada iteração. O mais comum é ser escolhido o elemento que provoca uma menor deterioração no valor da função objectivo de acordo com uma regra simples (para a heurística ser rápida) e razoável para o problema em causa (no sentido de se pretender uma solução de qualidade). Uma característica comum das heurísticas construtivas é serem "gulosas" (*greedy*): optam pelo melhor elemento no passo actual sem olhar às consequências para além deste.

As heurísticas construtivas têm duas qualidades relevantes que facilitam a aceitação da utilização de optimização nalgumas aplicações: podem incluir a tradução de procedimentos humanos para procedimentos computacionais e a sua correcta

[10] ou método aproximado por oposição a método exacto (método que garante a obtenção de uma solução óptima).

implementação é facilmente verificável. No entanto, estima-se que as soluções obtidas por heurísticas construtivas fiquem a 10-15% do valor óptimo.

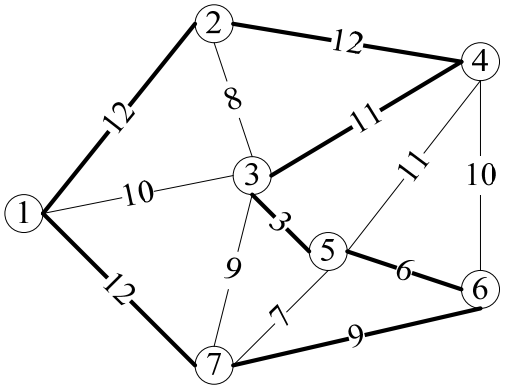
4.4.1 Heurística do vizinho mais próximo[11]

Como o nome indica, a heurística do vizinho mais próximo para o PCV, parte de um vértice e vai formando o circuito seleccionando o vértice mais próximo. Esta heurística baseia-se na representação de uma solução do PCV através de uma permutação. Assim, os elementos adicionados à solução são vértices. O algoritmo é dado de seguida.

```
// Algoritmo do vizinho mais próximo
Considerar um vértice arbitrariamente e o caminho formado apenas
         por esse vértice
Enquanto há vértices que não fazem parte do caminho
     Determinar o vértice mais próximo do último vértice adicionado
          ao caminho (de entre os que não fazem parte do caminho)
     Se não existe, terminar (heurística não obteve solução
          admissível)
     Se existe, incluí-lo no caminho
Formar o circuito, incluindo no caminho a aresta que liga o último
         vértice considerado ao primeiro
```

Considere-se a instância do PCV da Figura 4.1. Se se aplicar a heurística do vizinho mais próximo partindo o vértice 1, não é obtida nenhuma solução admissível (1-3-5-6-7 e algoritmo pára porque não há nenhum vértice não visitado a partir de 7), o que é consequência da heurística, em cada passo, escolher o melhor elemento sem avaliar as consequências futuras dessa escolha.

Na Figura 4.5 apresenta-se a solução obtida por aplicação da heurística do vizinho mais próximo partindo do vértice 3. A solução (admissível) obtida (3-5-6-7-1-2-4-3) tem um custo total de 65.



[11] Visualização da heurística do vizinho mais próximo em http://www-e.uni-magdeburg.de/mertens/TSP/TSP.html

Figura 4.5. Solução dada pela heurística do vizinho mais próximo (partindo de 3).

Considere-se agora a instância da Tabela 4.1. Aplicando a heurística do vizinho mais próximo partindo do vértice 1, como se pode verificar na Tabela 4.4, a solução obtida é 1-3-2-5-4-1 com distância total de 36.

c_{ij}	1	2	3	4	5
1	-	6	**3**	13	11
2	6	-	7	10	**9**
3	3	**7**	-	8	15
4	**13**	10	8	-	4
5	11	9	15	**4**	-

Tabela 4.4. Solução da heurística de menor custo.

Ao contrário do que é comum, existe uma garantia de qualidade para uma solução obtida pela heurística que consiste em aplicar a heurística do vizinho mais próximo n vezes com início em cada um dos n vértices e escolher a melhor solução. Para tal, as distâncias têm de ser todas positivas, a matriz das distâncias ser simétrica e satisfazer a desigualdade triangular, isto é $c_{ij} \leq c_{ik} + c_{kj}, \forall i, \forall j, \forall k$. Para essa heurística e com esses pressupostos, representando z_H como a distância dada pela heurística e por z_{OPT} a distância óptima, verifica-se

$$\frac{z_H}{z_{OPT}} \leq \frac{1}{2}(1 + log_2\, n)$$

A extensão da heurística do vizinho mais próximo para o PCV assimétrico é directa.

4.4.2 Heurística da aresta com menor custo

A heurística construtiva da aresta de menor custo para o PCV baseia-se em, em cada iteração, considerar a aresta que tem menor custo (das que ainda não foram consideradas) que não torna a solução não admissível. Assim, uma aresta é candidata a ser adicionada à solução se i) não fizer parte da solução, ii) a sua adição não implicar que um (ou dois) vértice(s) fique(m) com grau 3, iii) a sua adição não criar um subcircuito. Note-se que numa solução do problema do caixeiro viajante, todos os vértices têm grau 2 (cada vértice está ligada a outros dois) e não existem subcircuitos (só há um circuito que é o que inclui todos os vértices uma e uma só vez).

O algoritmo é dado de seguida.

```
// Algoritmo da aresta de menor custo
Enquanto circuito não está completo ou há arestas para considerar
     Seleccionar a aresta com menor custo das ainda não
          consideradas
          Se adição de aresta não implica a formação de um
          subcircuito nem vértices com grau 3, adicionar aresta à
          solução
Se arestas seleccionadas não formam circuito, heurística não obteve
     solução
```

Para a instância do PCV da Figura 4.1, a aplicação da aresta de menor custo resulta na consideração dos arcos pela ordem dada na Tabela 4.5, onde também é apresentada a decisão relativa a cada aresta. A aplicação desta heurística nesta instância não resulta numa solução admissível, como se pode constatar pela Figura 4.6.

Iteração	Aresta	Decisão
1	3-5	Incluída na solução
2	5-6	Incluída na solução
3	5-7	Excluída - vértice 5 ficaria com grau 3
4	2-3	Incluída na solução
5	6-7	Incluída na solução
6	3-7	Excluída – formaria subcircuito (3-5-6-7-3) e vértice 3 ficaria com grau 3
7	4-6	Excluída - vértice 6 ficaria com grau 3
8	1-3	Excluída - vértice 3 ficaria com grau 3
9	3-4	Excluída - vértice 3 ficaria com grau 3
10	4-5	Excluída - vértice 5 ficaria com grau 3
11	1-2	Incluída na solução
12	1-7	Excluída - formaria subcircuito
13	2-4	Excluída - vértice 2 ficaria com grau 3

Tabela 4.5. Aplicação da heurística de menor custo.

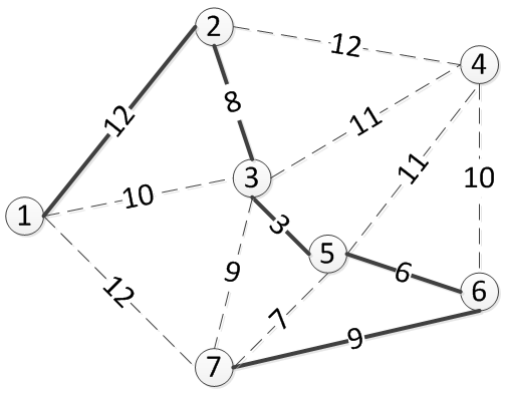


Figura 4.6. Solução dada pela heurística de menor custo.

Considere-se agora a instância da Tabela 4.1. Aplicando a heurística da aresta de menor custo, como se pode verificar nas Tabela 4.6 e Tabela 4.7, a solução obtida é 1-3-4-5-2-1 com distância total de 30.

c_{ij}	1	2	3	4	5
1	-	**6**	**3**	13	11
2		-	*7*	10	**9**
3			-	**8**	15
4				-	**4**
5					-

Tabela 4.6. Instância do PCV simétrico representada por uma matriz.

Iteração	Aresta	Decisão
1	1-3	Incluída na solução
2	4-5	Incluída na solução
3	1-2	Incluída na solução
4	2-3	Excluída - formaria subcircuito (1-2-3)
5	3-4	Incluída na solução
6	2-5	Incluída na solução

Tabela 4.7. Aplicação da heurística da aresta de menor custo.

A extensão da heurística da aresta de menor custo para o PCV assimétrico é quase directa, sendo apenas necessário alterar a exclusão das arestas que implicam vértices com grau 3 para a exclusão de arestas que implicam grau de entrada 2 ou grau de saída 2. Numa solução admissível do PCV simétrico, todos os nodos têm grau de entrada 1 e grau de saída 1.

4.4.3 Heurística de inserção do vértice mais próximo[12]

Na heurística de inserção do vértice mais próximo, a solução inicial é um subcircuito formado pela duplicação da aresta de menor custo. Em cada iteração é inserido um vértice no subcircuito (implicando a remoção de uma aresta e a adição de duas novas arestas) até não ser possível adicionar mais vértices por um circuito (completo) ter sido obtido ou por não existirem arestas que o permitam. O algoritmo é dado de seguida.

[12] Visualização de heurísticas de inserção em http://www-e.uni-magdeburg.de/mertens/TSP/TSP.html

```
// Algoritmo de inserção
Obter o subcircuito i-j-i em que ij é a aresta com menor custo
Enquanto circuito não está completo
     // Seleccionar vértice a inserir
     Seleccionar o vértice k que não faz parte do circuito e que
          tem a menor distância ao subcircuito (a distância do
          vértice ao subcircuito é dada pela menor distância entre o
          vértice e um vértice do circuito)
     // Seleccionar local a inserir
     Seleccionar aresta do circuito ij com menor valor de c_ik + c_kj -
          c_ij
     Actualizar circuito por remoção da aresta ij e adição das
          arestas ik e kj
```

Considere-se a instância da Tabela 4.1. O circuito inicial é 1-3-1 com distância 6. As distâncias dos vértices que não fazem parte do circuito ao circuito são:

$$d_2 = \min\{6,7\} = 6$$
$$d_4 = \min\{13,8\} = 8$$
$$d_5 = \min\{11,15\} = 11$$

Assim, o vértice escolhido para fazer parte do circuito é o vértice 2 porque tem a menor distância. Uma das arestas 1-3 é removida da solução e são inseridas as arestas 1-2 e 2-3. A variação da distância é $-3 + 6 + 7 = 10$ e a solução passa a valer $6 + 10 = 16$.

O subcircuito passa a ser 1-2-3-1. Actualizando as distâncias:

$$d_4 = \min\{13,8,10\} = 8$$
$$d_5 = \min\{11,15,9\} = 9$$

O vértice escolhido para fazer parte do circuito é o vértice 4 porque tem a menor distância. O vértice 4 pode ser inserido entre os vértices 1 e 2, ou 2 e 3, ou 3 e 1. As variações de custo para cada uma das situações são

$$\Delta_{12}^4 = 13 + 10 - 6 = 17$$
$$\Delta_{23}^4 = 10 + 8 - 7 = 11$$
$$\Delta_{13}^4 = 14 + 8 - 3 = 19$$

Por a variação de custo ser menor, o vértice 4 é inserido entre 2 e 3, passando a solução a ser 1-2-4-3-1 com valor $16 + 11 = 27$.

O vértice 5 pode ser inserido entre 1-2, 2-4, 4-3 e 3-1.

$$\Delta_{12}^5 = 11 + 9 - 6 = 14$$
$$\Delta_{24}^5 = 9 + 4 - 10 = 3$$
$$\Delta_{43}^5 = 15 + 4 - 8 = 11$$
$$\Delta_{13}^5 = 11 + 15 - 3 = 23$$

Por a variação de custo ser menor, o vértice 5 é inserido entre 2 e 4, passando a solução a ser 1-2-5-4-3-1 com valor $27 + 3 = 30$.

Se a instância fosse assimétrica, seria necessário calcular a variação de custo da inserção de cada vértice em cada um dos sentidos.

4.4.4 *Outras heurísticas de inserção

Outras heurísticas de inserção podem ser obtidas alterando a forma como é seleccionado o subcircuito inicial e/ou a forma de selecção do vértice e do local de inserção.

Por exemplo, na heurística de inserção do vértice mais afastado, o vértice seleccionado, como o nome indica, é o que tem a maior distância ao circuito.

Na heurística da inserção mais barata, são calculadas todas as combinações entre arestas do subcircuito e vértices que não fazem parte do subcircuito, seleccionando-se a que tem menor variação.

Na heurística de inserção no invólucro convexo, que apenas faz sentido aplicar em instâncias do PCV euclidianas (e.g. a distância entre dois vértices é a distância euclidiana – que se obtém com base nas coordenadas dos vértices), o subcircuito inicial é obtido com base no invólucro convexo.

O invólucro convexo de um conjunto de pontos X é o conjunto convexo mais pequeno que contém X, tal como ilustrado na Figura 4.7. Na heurística de inserção no invólucro convexo o subcircuito inicial é formado pelos vértices que fazem parte do invólucro convexo. A selecção do vértice k a inserir e da localização da inserção (entre i e j) é feita com base em $\frac{c_{ik}+c_{kj}}{c_{ij}}$.

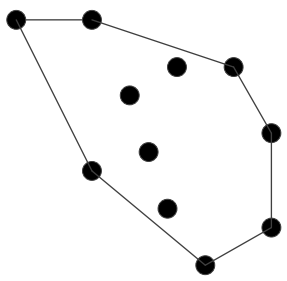


Figura 4.7. Ilustração do conceito de invólucro convexo.

4.5 Heurísticas específicas

Na secção anterior foram apresentadas heurísticas com aspectos específicos do PCV mas que seguem o princípio geral das heurísticas construtivas. Nesta secção apresentam-se duas heurísticas que são totalmente específicas do PCV (embora os mesmos princípios possam estar presentes em heurísticas para problemas relacionados como o problema do caixeiro viajante múltiplo – em que se podem construir vários circuitos - ou problemas de encaminhamento de veículos – o mais simples dos quais se caracteriza por os vértices terem procuras e os veículos capacidades). Refere-se o facto de existirem outras heurísticas específicas, como a Or-opt, Lin-Kernighan, GENI e GENIUS, mais elaboradas e que produzem resultados, em geral, de melhor qualidade. Por exemplo, estima-se que a heurística de Lin-Kernighan obtenha soluções a cerca de 1-2% do valor óptimo.

4.5.1 Heurística de Clarke e Wright

A heurística de Clarke-Wright (também designada por heurística das poupanças) parte de uma solução não admissível que consiste num conjunto de $n-1$ subcircuitos e, em cada iteração, funde dois subcircuitos. Os subcircuitos iniciais são definidos por um vértice central (escolhido arbitrariamente) e cada um dos restantes vértices. A fusão de dois subcircuitos consiste em remover duas arestas, uma de cada subcircuito, que incidem no vértice central e adicionar uma aresta que une os dois vértices (que não o central) que ficam com grau 1. No máximo (quando os dois circuitos têm mais de dois vértices), existem quatro formas de unir dois subcircuitos.

Em cada iteração é calculada a variação de custo (inverso da poupança) das possíveis fusões de cada par de subcircuitos e é efectuada a fusão com a menor variação de custo (maior poupança).

A poupança de unir dois subcircuitos em que i faz parte de um subcircuito e j faz parte do outro subcircuito e ambos estão ligados ao nodo central é

$$s_{ij} = \begin{cases} c_{ic} + c_{cj} - c_{ij}, \ se\ a\ aresta\ ij\ é\ admissível \\ -\infty, se\ a\ aresta\ ij\ não\ é\ admissível \end{cases}$$

```
// Algoritmo de Clarke e Wright
Seleccionar arbitrariamente um nodo central c e adicionar à solução
         as arestas entre c e cada um dos restantes nodos duas
         vezes
Enquanto não existir apenas um circuito
     Para cada par de subcircuitos c-i1-...-j1-c e c-i2-...-j2-c,
         considerar a poupança para
         remoção das arestas c-i1 e c-i2 e adição da aresta i1-i2
         remoção das arestas c-i1 e c-j2 e adição da aresta i1-j2
         remoção das arestas c-j1 e c-i2 e adição da aresta j1-i2
         remoção das arestas c-j1 e c-j2 e adição da aresta j1-j2
     Fundir o par de subcircuitos com maior poupança
```

O cálculo inicial da matriz das poupanças torna o algoritmo mais eficiente.

Considere-se um exemplo da Tabela 4.8 com cinco vértices. O vértice 1 é escolhido arbitrariamente para vértice central.

Os subcircuitos iniciais são 1-2-1 (custo 12), 1-3-1 (custo 6), 1-4-1 (custo 26) e 1-5-1 (custo 22) com custo total de 66.

c_{ij}	1	2	3	4	5
1	-	6	3	13	11
2		-	7	10	9
3			-	8	15
4				-	4
5					-

Tabela 4.8. Instância do PCV simétrica representada por uma matriz.

As poupanças são dadas na Tabela 4.9. Por exemplo, a poupança na célula 2-3 corresponde a fundir os subcircuitos 1-2-1 e 1-3-1. Tal implica a remoção das arestas 1-2 e 1-3 e a adição da aresta 2-3. A poupança é $6 + 3 - 7 = 2$. Note-se que se os subcircuitos 1-3-1 e 1-5-1 forem fundidos, a solução obtida tem um custo maior do que a actual.

s_{ij}	1	2	3	4	5
1	-	-	-	-	-
2	-	-	2	9	8
3	-		-	8	-1
4	-			-	20
5	-				-

Tabela 4.9. Poupanças do exemplo.

A fusão escolhida é a dos subcircuitos 1-4-1 e 1-5-1, passando a fazer parte da solução o subcircuito 1-4-5-1. O custo da nova solução é $66 - 20 = 46$.

As possíveis fusões são agora 1-2-1 com 1-3-1, 1-2-1 com 1-4-5-1 e 1-3-1 com 1-4-5-1. O maior valor da tabela das poupanças (excluindo células de pares de vértices no mesmo subcircuito. i.e. a célula 4-5) é 9 e corresponde a fundir os circuitos 1-2-1 e 1-4-5-1 removendo as arestas 1-2 e 1-4 e adicionando a aresta 2-4. O subcircuito 1-2-4-5-1. O custo da nova solução é $46 - 9 = 37$. Resta considerar a fusão do circuito 1-3-1 com o circuito 1-2-4-5-1 que pode ser feita de duas formas, adicionando a aresta 2-3 ou adicionando a aresta 3-5. Sendo a poupança maior no primeiro caso, o circuito obtido é 1-3-2-4-5-1 com custo $37 - 2 = 35$.

Para instâncias assimétricas, seria necessário ter em conta a orientação dos subcircuitos.

4.5.2 Heurística da árvore de suporte de custo mínimo

O problema da árvore de suporte de custo mínimo define-se numa rede não orientada em que existe um custo associado a cada aresta. Uma árvore de suporte é um conjunto de $n - 1$ arestas (em que n é o número de vértices da rede) que não forma (sub)circuitos. É possível determinar uma solução óptima para este problema através da aplicação, entre outros, do algoritmo de Kruskal que consiste em ordenar as arestas por ordem crescente de custo e considerando um arco de cada vez, testar se a sua inclusão na solução forma um (sub)circuito. Se a inclusão do arco não formar um (sub)circuito, o arco é incluído na árvore de suporte; caso contrário, passa-se para a aresta seguinte. Note-se que não é frequente um algoritmo construtivo, como o agora descrito, garantir a obtenção de uma solução óptima.

Dada a existência de algoritmos muito eficientes e que obtêm a solução óptima para o problema da árvore de suporte de custo mínimo (como o de Kruskal agora referido) e a semelhança com o problema do caixeiro viajante – em ambos os problemas pretende-se minimizar o custo de um conjunto de arestas, embora este conjunto defina

objectos diferentes), parece razoável a estratégia de primeiro obter uma árvore de suporte de custo mínimo e em seguida transformá-la numa solução para o problema do caixeiro viajante.

Assim, num primeiro passo, obtém-se a árvore de suporte de custo mínimo. Num segundo passo, cada aresta da árvore de suporte de custo mínimo é duplicada. Em seguida, gera-se um circuito Euleriano[13] que existe sempre por o número de arestas ser par de acordo com o teorema de Euler. Este circuito é transformado numa solução do PCV (um circuito Hamiltoniano): percorrendo-se o circuito Euleriano, os vértices que ainda não foram visitados são adicionados ao circuito Hamiltoniano.

```
// Algoritmo da heurística da árvore de suporte de custo mínimo
Obter árvore de suporte de custo mínimo
Duplicar todas as arestas da árvore de suporte de custo mínimo
Obter um circuito Euleriano
Transformar o circuito Euleriano num circuito Hamiltoniano
         percorrendo o circuito Euleriano sem repetição de arcos e
         adicionando ao circuito Hamiltoniano, pela mesma ordem, os
         vértices que ainda dele não fazem parte
```

A título exemplificativo, considere-se a instância da Tabela 4.10.

c_{ij}	1	2	3	4	5
1	-	**6**	**3**	**8**	11
2		-	7	10	9
3			-	9	11
4				-	**4**
5					-

Tabela 4.10. Instância do PCV simétrico.

A aplicação do algoritmo de Kruskal é dada na Tabela 4.11 e resulta na árvore de suporte de custo mínimo 1-2, 1-3, 1-4, 4-5.

Iteração	Aresta	Decisão
1	1-3	Incluída na solução
2	4-5	Incluída na solução
3	1-2	Incluída na solução
4	2-3	Excluída - formaria subcircuito (1-2-3)
5	1-4	Incluída na solução

Tabela 4.11. Aplicação do algoritmo de Kruskal.

[13] Circuito Euleriano - de Euler (1707-1783) - é um circuito que inclui todos os arcos uma e uma só vez (os nodos podem ser repetidos). Teorema de Euler: um grafo não orientado tem um circuito Euleriano se e só se todos os vértices têm grau par.

O passo seguinte é duplicação das arestas. O circuito Euleriano, tal como representado na Figura 4.8, é 1-4-5-4-1-3-1-2-1. O circuito Hamiltoniano, representado pelos arcos a negro na Figura 4.9 é então 1-4-5-3-2-1.

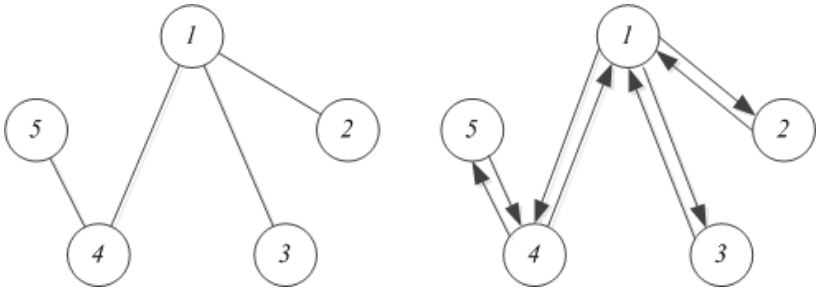


Figura 4.8. Árvore de suporte de custo mínimo e caminho euleriano.

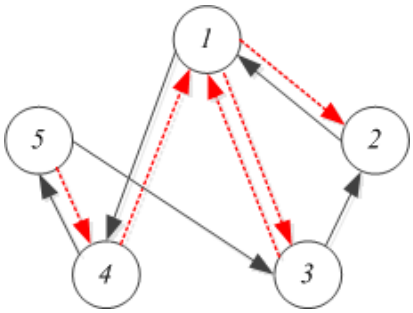


Figura 4.9. Construção da solução do PCV com a heurística da árvore de suporte de custo mínimo.

Um melhoramento desta heurística (a heurística de Christofides) obtém o circuito Euleriano através da resolução de um problema de emparelhamento perfeito de custo mínimo associado aos nodos com grau ímpar da árvore (e não por substituição das arestas por dois arcos de sentidos opostos).

4.6 Heurísticas de pesquisa local

O conceito fundamental de uma heurística de pesquisa local é o conceito de vizinhança. A vizinhança de uma solução é o conjunto de soluções que se podem obter a partir dela através de uma modificação elementar. Essencialmente, o que diferencia uma heurística de pesquisa local de outra para o mesmo problema é a estrutura de vizinhança.

Um algoritmo genérico de uma heurística de pesquisa local (para um problema de minimização) é o seguinte.

```
// Algoritmo de pesquisa local genérico
Obter uma solução inicial x (através de uma heurística construtiva,
         aleatoriamente, ou de outra forma)
// Melhorou é uma variável booleana
Melhorou=true
Enquanto Melhorou==true
     Avaliar todas as soluções vizinhas de x e guardar a melhor y
     Se f(y)<f(x), x=y
     Se não, Melhorou=false
```

O algoritmo apresentado é designado por máxima descida por que é escolhida a melhor solução vizinha. Alternativamente, pode ser escolhida a primeira solução que é melhor do que a actual. Uma variante estocástica do algoritmo de pesquisa local consiste em fazer uma amostra da vizinhança de forma aleatória (e, tal como na pesquisa determinística, escolher a melhor vizinha ou a primeira melhor do que a actual, de entre as consideradas).

No final da aplicação do algoritmo de pesquisa local (determinístico) a solução obtida é um óptimo local (por oposição a global) para a estrutura de vizinhança utilizada.

4.6.1 Troca de duas arestas (2-Opt)[14]

A estrutura de vizinhança de troca de duas arestas para o PCV, também conhecida por 2-Opt, consiste em remover duas arestas não adjacentes do circuito e reconstruir o circuito com outras duas arestas. Na Figura 4.10 mostram-se as cinco soluções vizinhas (de acordo com a estrutura de vizinhança de troca de duas arestas) de uma solução de um PCV com cinco vértices. Note-se que, após removidas duas arestas só há duas arestas que, ao serem inseridas, fazem com que se forme um circuito.

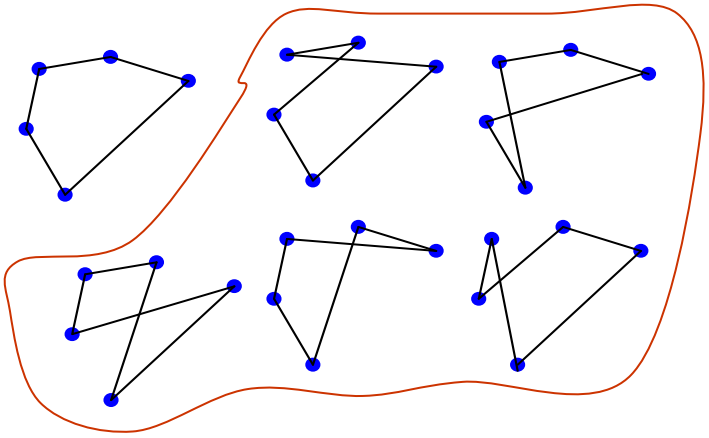


Figura 4.10. As cinco soluções vizinhas de uma solução num PCV com cinco vértices.

[14] Visualização da heurística 2-Opt em http://www-e.uni-magdeburg.de/mertens/TSP/TSP.html

Se se representar de uma solução do PCV como uma permutação (sequência de números de 1 a n sem repetições), obtém-se uma solução vizinha 2-Opt pode ao inverter-se uma subsequência (de comprimento até $n-2$). Esta forma expedita de obter soluções vizinhas 2-Opt é agora exemplificada com a instância da Tabela 4.12.

c_{ij}	1	2	3	4	5
1	-	6	3	13	11
2	6	-	7	10	9
3	3	7	-	8	15
4	13	10	8	-	4
5	11	9	15	4	-

Tabela 4.12. Instância do PCV simétrico.

Aplicando a heurística do vizinho mais próximo partindo do vértice 1 obtém-se a solução 1-3-2-5-4-1 com distância total de 36. A primeira iteração da heurística da troca de duas arestas é dada nas Tabela 4.13, assumindo uma estratégia de pesquisa de máxima descida (determinística) a solução actual da iteração seguinte é a solução 1-3-4-5-2 (solução vizinha 4) por ter o menor custo e ser melhor do que a solução actual.

						Custo	Arestas removidas	Arestas inseridas	Subsequência invertida
Solução actual	1	3	2	5	4	36	-	-	-
Solução vizinha 1	1	**2**	**3**	5	4	45	1-3 e 2-5	1-2 e 3-5	3-2
Solução vizinha 2	1	**5**	**2**	**3**	4	48	1-3 e 5-4	1-5 e 3-4	3-2-5
Solução vizinha 3	1	3	**5**	**2**	4	50	3-2 e 5-4	3-5 e 2-4	2-5
Solução vizinha 4	1	3	**4**	**5**	**2**	30	3-2 e 4-1	3-4 e 1-2	2-5-4
Solução vizinha 5	1	3	2	**4**	**5**	35	2-5 e 4-1	2-4 e 1-5	5-4

Tabela 4.13. Primeira iteração da heurística da troca de duas arestas.

Na Tabela 4.14 é dada a informação relativa à segunda iteração da heurística da troca de duas arestas. Como todas as soluções que se obtêm por troca de duas arestas são piores (ou iguais) do que solução a actual, a solução actual é uma solução óptima local (relativamente à estrutura de vizinhança de troca de duas arestas) e o algoritmo termina.

						Custo
Solução actual	1	3	4	5	2	30
Solução vizinha 1	1	**4**	**3**	5	2	51
Solução vizinha 2	1	**5**	**4**	**3**	2	36
Solução vizinha 3	1	3	**5**	**4**	2	38
Solução vizinha 4	1	3	**2**	**5**	**4**	36
Solução vizinha 5	1	3	4	**2**	**5**	41

Tabela 4.14. Segunda iteração da heurística da troca de duas arestas.

4.6.2 Troca de três arestas (3-Opt)

Na estrutura de vizinhança de trocas de três arestas (3-Opt), a vizinhança de uma solução é o conjunto de soluções que se obtém ao remover três arestas não adjacentes do circuito e a reconstruir o circuito com outras três arestas (em que pelo menos uma é diferente das removidas). Na Figura 4.11 estão representadas as sete formas de inserir arestas depois de definidas as três arestas que são removidas de uma solução. Quatro vizinhas têm três arestas diferentes da solução actual e três soluções têm apenas duas arestas diferentes da solução actual.

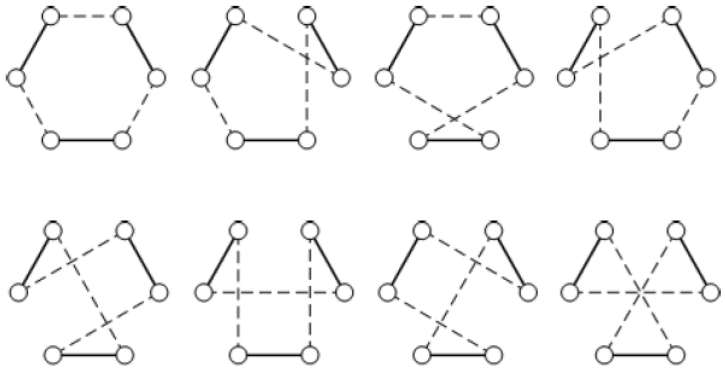


Figura 4.11. Solução actual (canto superior esquerdo) e sete soluções vizinhas 3-Opt obtidas pela remoção das três arestas a tracejado na solução actual.

Estima-se que a heurística 3-Opt obtenha soluções a 3-4% do valor óptimo.

4.6.3 Troca de dois vértices

Na estrutura de vizinhança de troca de dois vértices, como o nome indica, a troca de posição de dois vértices de uma solução. Por exemplo, a solução 1-2-3-4-5 terá como vizinhas as soluções 2-1-3-4-5, 3-1-2-4-5, 4-1-2-3-5, ... Numa instância simétrica com n vértices o número de vizinhas é $n(n-1)/2$.

Na Figura 4.12 é apresentado um exemplo de uma solução actual 1-2-3-4-5-6-7 (custo 69) e da solução vizinha que se obtém por troca das cidades 3 e 4, ou seja 1-2-4-3-5-6-7 (custo 65).

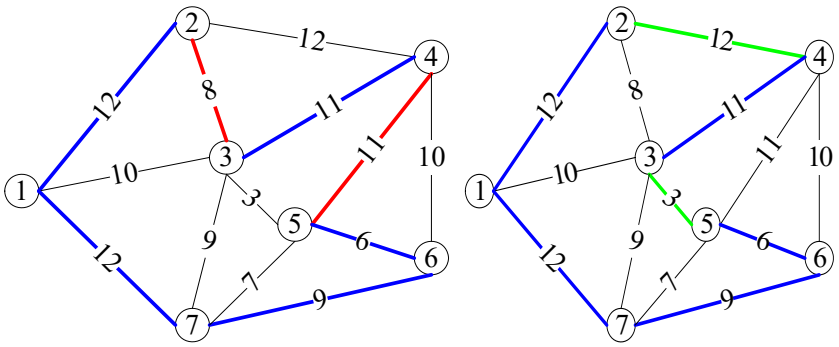


Figura 4.12. Exemplo de solução vizinha por troca de dois vértices.

4.7 *Meta-heurísticas[15]

A ideia fundamental das meta-heurísticas é ter um procedimento a um nível acima (meta) da heurística (em geral de pesquisa local) que permita que o encontrar de soluções de qualidade (em particular óptimos locais) uma solução óptima local não termine com a exploração do espaço de soluções. Cada meta-heurística tem os princípios que a caracterizam mas também componentes específicas do problema, usualmente incorporados na pesquisa local.

4.7.1 GRASP (Greedy Adaptive Search Procedure)

Numa meta-heurística GRASP, em cada iteração é aplicada uma heurística construtiva (aleatorizada) seguida de pesquisa local. Na sua versão mais básica, as iterações são independentes entre si, podendo o GRASP ser visto como uma heurística de pesquisa local com múltiplos inícios (multi-start local search - pesquisa local aplicada várias vezes mas com soluções iniciais diferentes).

Em cada iteração da heurística construtiva não é necessariamente seleccionado o melhor elemento para entrar na solução (como nas heurísticas construtivas "greedy") mas é escolhido um elemento aleatoriamente de entre os que pertencem a uma lista

[15] Na continuidade das heurísticas apresentadas, descrevem-se duas meta-heurísticas cuja principal característica é a relação entre componentes construtivas e de vizinhanças (GRASP e VND). A pesquisa tabu introduz o conceito de memória. Outros aspectos importantes em meta-heurísticas como a aleatoriedade (usado no GRASP e no VNS, e que tem um papel na meta-heurística clássica pesquisa por arrefecimento simulado aqui não abordada) e a utilização de populações (como nos algoritmos genéticos ou no pesquisa por dispersão) são deixados para um aprofundamento do tópico.

restrita de candidatos (RCL – restricted candidate list) que contém os melhores elementos.

Um elemento j faz parte da RCL se o seu custo não estiver acima de uma determinada proporção da amplitude entre o maior e menor custo (em minimização): $c_j \leq c_{min} + \alpha(c_{max} - c_{min})$. Desta forma garante-se que os elementos com o melhor valor nunca são excluídos.

O parâmetro α, $\alpha \in [0,1]$, regula o tamanho da RCL e o grau de aleatoriedade da selecção. Se $\alpha = 0$, o GRASP é equivalente a uma heurística construtiva seguida de pesquisa local. Se $\alpha = 1$, o GRASP é próximo de obter uma solução aleatória para inicializar uma pesquisa local.

O algoritmo é dado de seguida.

```
Enquanto critério de paragem não for atingido
     // Algoritmo construtivo
     Começar com solução vazia
     Construir conjunto de elementos candidatos a entrar na solução
          e avaliá-los
     Repetir até não haver elementos candidatos
          Definir uma Lista Restrita de Candidatos (RCL)
          Seleccionar aleatoriamente um elemento da RCL
          Adicionar o elemento seleccionado à solução
          Definir o conjunto de elementos candidatos a entrar na
          solução  e avaliá-los
     // Pesquisa local
     Partindo da solução obtida, aplicar pesquisa local
```

Qualquer das heurísticas construtivas e das heurísticas de pesquisa local apresentadas para o PCV podem ser usadas como componentes de uma meta-heurística GRASP.

4.7.2 Vizinhanças variáveis

Nesta subsecção descrevem-se duas meta-heurísticas próximas da pesquisa local: descida em vizinhanças variáveis (VND - variable neighborhood descent) e pesquisa em vizinhanças variáveis (VNS - variable neighborhood search).

O conceito fundamental destas abordagens é usar mais do que uma vizinhança. As vizinhanças são ordenadas das mais pequenas (menos modificações na solução actual e pesquisadas de forma mais eficiente) para as maiores.

No VND, começa-se por efectuar uma pesquisa local de acordo com uma estrutura de vizinhança. Quando é obtido um óptimo local para a primeira estrutura de vizinhança, passa-se a usar uma segunda estrutura de vizinhança. Se há uma actualização da solução actual, regressa-se à estrutura de vizinhança actual.

```
Obter solução inicial
Vizinhança actual é a primeira
Enquanto a vizinhança actual não for a última
     Escolher melhor vizinha de acordo com a vizinhança actual
     Se vizinha é melhor do que actual, vizinhança actual passa a
          ser a primeira
     Se não, vizinhança actual passa a ser a vizinhança seguinte
```

No VNS apenas a primeira estrutura de vizinhança é usada para efectuar pesquisa local. As restantes são usadas para seleccionar (de forma aleatória) uma solução a partir da qual se aplica a pesquisa local.

No caso do PCV, as estruturas de vizinhança 2-Opt e 3-Opt, dada a sua estrutura hierárquica, são bons pontos de partida para a aplicação de VND e VNS.

4.7.3 Pesquisa tabu

Na pesquisa tabu, ao contrário do que acontece na pesquisa local, uma solução vizinha pode ser seleccionada para solução actual mesmo sendo pior do que esta. Na sua versão mais simples, a pesquisa tabu inclui um mecanismo de memória (a lista tabu) que evita a entrada em ciclo, permitindo que a pesquisa seja diversificada.

Na lista tabu são guardados os movimentos mais recentes, em que um movimento é a alteração efectuada a uma solução. Por exemplo, na vizinhança 2-Opt um movimento corresponde a duas arestas que são removidas.

Assim, em cada iteração da pesquisa tabu, é seleccionada a melhor solução vizinha de entre as que (i) não são tabu ou (ii) são tabu mas respeitam critério de aspiração (por exemplo, serem melhores do que a solução incumbente, i.e., a melhor solução encontrada até ao momento).

O algoritmo é dado de seguida.

```
Obter uma solução inicial
Repetir
     Gerar vizinhança da solução actual
     Escolher melhor solução da vizinhança tal que: solução não é
          tabu ou é tabu mas satisfaz o critério de aspiração
     Actualizar lista tabu por remoção do movimento mais antigo e
          inserção do movimento mais recente
     Se a solução actual é melhor do que a solução incumbente,
          actualizar solução incumbente
Até ser atingido um número de iterações sem melhoria da incumbente
          (ou um número de iterações limite ou um tempo limite)
```

4.8 *Modelos de programação inteira

Excepto quando explicitamente indicado, nesta secção, consideram-se instâncias do PVC assimétrico.

Todos os modelos usam as variáveis de decisão x_{ij} que definem um circuito

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

e todos os modelos usam as seguintes restrições que forçam a que todos os nodos tenham grau de entrada 1 (um arco a entrar) e um grau de saída 1 (um arco a sair):

$$\sum_{j:ij \in A} x_{ij} = 1\,, \forall i \in N$$
$$\sum_{j:ji \in A} x_{ji} = 1\,, \forall i \in N$$
$$x_{ij} \in \{0,1\}, \forall ij \in A$$

A função objectivo é também a mesma em todos os modelos:

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

4.8.1 Dantzig-Fulkerson-Johnson

No modelo de Dantzig-Fulkerson-Johnson consideram-se adicionalmente as restrições que eliminam subcircuitos:

$$\sum_{ij \in A(S)} x_{ij} \leq |S| - 1, \forall S \subset N\,, |S| \geq 2$$

Para um conjunto de nodos S, $A(S)$ é o conjunto dos arcos que têm ambas as extremidade em nodos de S. O número de nodos de S (o cardinal do conjunto S) é representado por $|S|$.

Existe uma restrição de eliminação de subcircuitos para cada subconjunto de nodos S. A restrição para um subconjunto de nodos S indica que o número de arcos entre nodos de S tem de ser menor do que o número de nodos de S. Assim, um conjunto de arcos que forme um circuito envolvendo todos os nodos de S não é admissível. Por exemplo, numa instância com seis nodos numa rede completa, $N = \{1,2,3,4,5,6\}$, ao conjunto de nodos $S = \{1,2,3\}$, corresponde $A(S) = \{12,13,21,23,31,32\}$ e a restrição que elimina o subcircuito 1-2-3 e o subciruito 3-2-1 é $x_{12} + x_{13} + x_{21} + x_{23} + x_{31} + x_{32} \leq 2$.

Dado que o número de subconjuntos de nodos cresce exponencialmente com o aumento da rede, a resolução directa do modelo de Dantzig-Fulkerson-Johnson não é praticável em instâncias de alguma dimensão. Usualmente é aplicado o algoritmo seguinte:

```
Resolver o problema sem as restrições de eliminação de subcircuitos
Enquanto a solução contiver subcircuitos
     Identificar os nodos dos subcircuitos
     Para cada subcircuito, definir S como o conjunto dos nodos do
          subcircuito e adicionar a restrição correspondente ao
          problema
     Resolver problema obtido
```

A título exemplificativo, considere-se a instância do problema do caixeiro viajante definida pela matriz de custos.

c_{ij}	*1*	*2*	*3*	*4*	*5*	*6*
1	100	4	5	4	8	6
2	5	100	3	10	6	9
3	5	10	100	1	3	10
4	3	5	1	100	8	6
5	10	10	10	8	100	7
6	1	8	2	3	5	100

O modelo sem restrições de eliminação de subcircuitos é

$$Min\ z = 4x_{11} + 5x_{12} + \cdots + 5x_{65}$$
$$sujeito\ a:$$
$$x_{12} + x_{13} + \cdots + x_{16} = 1$$
$$x_{21} + x_{23} + \cdots + x_{26} = 1$$
$$\ldots$$
$$x_{61} + x_{62} + \cdots + x_{65} = 1$$
$$x_{21} + x_{31} + \cdots + x_{61} = 1$$
$$\ldots$$
$$x_{61} + x_{26} + \cdots + x_{56} = 1$$
$$x_{ij} \in \{0,1\}, \forall ij \in A$$

Uma solução óptima deste modelo (sem as restrições de eliminação de subcircuitos) é $x_{12} = x_{25} = x_{56} = x_{61} = x_{34} = x_{43} = 1$ e todas as restantes variáveis com valor zero. O valor da solução é 20.

Esta solução tem dois subcircuitos: $1-2-5-6-1$ e $3-4-3$. Para eliminar cada um destes dois subcircuitos. adicionam-se as restrições (para $S = \{1,2,5,6\}$ e $S = \{3,4\}$):

$$x_{12} + x_{15} + x_{16} + x_{21} + x_{25} + x_{26} + x_{51} + x_{52} + x_{56} + x_{61} + x_{62} + x_{65} \leq 3$$
$$x_{34} + x_{43} \leq 1$$

A nova solução óptima tem valor 23 e é $x_{12} = x_{23} = x_{35} = x_{56} = x_{64} = x_{41} = 1$. Dado não ter subcircuitos, esta solução é óptima do problema original (em que se consideram todas as restrições de eliminação de subcircuitos).

4.8.2 Miller-Tucker-Zemlin

O modelo de Miller-Tucker-Zemlin difere na forma como são eliminados os subcircuitos. Neste modelo consideram-se variáveis adicionais associadas à sequência pela qual as cidades são visitadas. A primeira cidade da sequência é a cidade 1. As variáveis contínuas (u) definem a sequência

$$u_i - posição\ em\ que\ é\ visitada\ a\ cidade\ i,\ i = 2, \dots, n$$

Tendo em conta que $x_{ij} = 1$ implica que $u_j \geq u_i + 1$ (j é visitado depois de i), o grupo de restrições

$$u_j \geq u_i + 1 - n(1 - x_{ij}), \forall ij \in A, i \neq 1, j \neq 1$$

$$u_i \geq 0, i = 2, \dots, n$$

garante a eliminação de subcircuitos.

Note-se que $x_{ij} = 1$ implica $u_j - u_i \geq 1$, i.e., o nodo j é forçado a estar depois do nodo i e que $x_{ij} = 0$ implica $u_j - u_i \geq -n + 1$, i.e., a restrição é redundante.

Para o exemplo introduzido anteriormente, uma solução óptima com valor 23 é $x_{12} = x_{25} = x_{56} = x_{63} = x_{34} = x_{41} = 1$ e $u_2 = 0,\ u_3 = 3,\ u_4 = 4,\ u_5 = 1, u_6 = 2$.

A grande vantagem deste modelo em relação ao anterior é o número de restrições de eliminação de subcircuitos ser muito menor, sendo próximo do número de arcos. Tal permite a resolução directa (não iterativa) do problema.

4.8.3 Fluxos

Apresentam-se dois modelos de fluxos, um modelo agregado e um modelo desagregado.

O modelo de fluxos (agregados) tem como e as variáveis de decisão adicionais

$$y_{ij} - fluxo\ no\ arco\ ij, \forall ij \in A$$

e as restrições adicionais

$$\sum_{1j \in A} y_{1j} = n - 1$$

$$\sum_{j:ij \in A} y_{ij} - \sum_{j:ji \in A} y_{ji} = -1\ , \forall i \in N \setminus \{1\}$$

$$y_{ij} \leq (n-1)x_{ij}, \forall ij \in A$$

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

$$y_{ij} \geq 0, ij \in A$$

O primeiro e segundo grupos de restrições forçam a que saia uma unidade de fluxo do nodo 1 para cada um dos nodos restantes. O terceiro grupo de restrições garante que se existe fluxo num arco, este arco tem de fazer parte do circuito (que é definido pelas variáveis x_{ij}).

No modelo de fluxo desagregado, as variáveis y_{ij} são desagregadas por destino do fluxo:

$$y_{ijk} - fluxo\ no\ arco\ ij\ com\ destino\ a\ k, \forall ij \in A,\ \forall k \in N \setminus \{1\}$$

As restrições adicionais são

$$\sum_{j:ij\in A} y_{ijk} - \sum_{j:ji\in A} y_{jik} = \begin{cases} 1\ se\ i = 1 \\ 0\ se\ i \neq k\ e\ i \neq 1, \forall i \in N, \forall k \\ -1\ se\ i = k \end{cases} \in N \setminus \{1\}$$

$$y_{ijk} \leq x_{ij}, \forall ij \in A, \forall k \in N \setminus \{1\}$$

$$y_{ij} \geq 0, ij \in A$$

O primeiro e segundo grupos de restrições forçam a que saia uma unidade de fluxo do nodo 1 para cada um dos nodos restantes. O terceiro grupo de restrições garante que se existe fluxo num arco, este arco tem de fazer parte do circuito (que é definido pelas variáveis x_{ij}).

4.8.4 Etapas

No modelo de etapas, as variáveis de decisão adicionais são

$$y_{ijt} = \begin{cases} 1, se\ arco\ ij\ é\ atravessado\ na\ etapa\ t \\ 0,\ caso\ contrário \end{cases}, \forall ij \in A, t = 1, \dots, n$$

As restrições adicionais são

$$\sum_{t\geq 2} \sum_{j:ij\in A} t y_{ijt} - \sum_{t} \sum_{j:ji\in A} t y_{jit} = 1,\ \forall i \in N \setminus \{1\}$$

$$\sum_{j:1j\in A} y_{1j1} = 1, \forall ij \in A$$

$$\sum_{j:j1\in A} y_{j1n} = 1, \forall ij \in A$$

$$x_{ij} = \sum_{t} y_{ijt}, \forall ij \in A$$

$$y_{ijt} \geq 0, \forall ij \in A, t = 1, \dots, n$$

O primeiro grupo de restrições indica que se um arco entra no nodo i (excluindo o nodo 1) na etapa t, então o arco da etapa seguinte sai de i (etapa $t+1$). O segundo e terceiro grupos de restrições asseguram que uma arco sai de 1 na primeira etapa e chega a 1 na última etapa. O quarto grupo de restrições assegura que se um arco é usado então tem de ser usado numa etapa e vice-versa.

4.8.5 Comparação entre modelos

Diferentes modelos têm comportamentos computacionais muito diferentes, o que deriva qualidade dos limites inferiores dados pela relaxação linear do modelo. O modelo de fluxos desagregado tem melhor comportamento computacional do que o modelo de fluxos agregado. O modelo de etapas tem um muito mau comportamento computacional.

4.9 *Problemas do caixeiro viajante com lucros

4.9.1 Definição

Nesta secção referem-se problemas relacionados com o problema do caixeiro viajante mas em que não é necessário visitar todos os vértices (pretende-se portanto um subcircuito) mas existe um lucro associado a visitar cada vértice.

A notação é a seguinte

- $G = (N, A)$ grafo orientado (instância simétrica) ou não orientado (instância assimétrica)
- n número de vértices
- c_{ij} distância em que se incorre ao atravessar o arco ij, $\forall ij \in A$
- $c_{ij} \geq 0, \forall ij \in A$
- p_i lucro associado a visitar o vértice $i, \forall i \in N$

Assume-se que o vértice 1 tem de ser visitado.

Existem dois critérios para avaliar a qualidade de uma solução: a distância percorrida e o lucro obtido. De forma natural, surge o problema bi-objectivo com os objectivos da minimização da distância do (sub)circuito e da maximização do lucro dos nodo do (sub)circuito.

No entanto, frequentemente, este problema bi-objectivo é tratado através de uma das três simplificações seguintes (que o transformam num problema de objectivo único):

- objectivo único é a soma ponderada dos dois objectivos (*profitable tour problem - PTP*);
- pretende-se maximizar o lucro com uma restrição de distância máxima (c_{max}) (*orienteering problem - OP*);
- pretende-se minimizar a distância com o lucro com uma restrição de lucro mínimo (p_{min}) (*prize collecting problem - PCP*).

4.9.2 Abordagens heurísticas

A heurística construtiva mais natural para problemas do caixeiro viajante com lucros é baseada na inserção de um vértice. Partindo do vértice 1, é inserido o vértice que mais aumenta a soma ponderada dos objectivos (PTP), mais aumenta o lucro (OP) ou o vizinho mais próximo (PCP). A heurística pára quando não é possível aumentar a soma ponderada dos dois objectivos (PTP), a inserção de qualquer vértice excede a distância máxima (OP) ou o lucro mínimo foi atingido (PCP). É de referir que deverá ser tido em conta que a inserção de um vértice pode ser feita em diferentes posições do (sub)circuito. Se o vértice k for inserido entre i e j, a variação da distância é $c_{ik} + c_{kj} - c_{ij}$ e a variação do lucro é p_k.

Uma heurística simétrica à agora descrita (poder-se-á chamar heurística destrutiva) consiste em obter uma solução do PCV (possivelmente heurística) e remover vértices iterativamente tendo em conta a variação da distância e do lucro de forma similar à heurística construtiva.

Dado que uma solução de um problema do problema do caixeiro viajante com lucros não tem um número de vértices fixo (ao contrário de uma solução para o problema do caixeiro viajante), as duas operações descritas (inserção e remoção) podem ser usadas para definir (sub)vizinhanças. Outra (sub)vizinhança consiste na substituição de um vértice do (sub)circuito por um vértice que não está no (sub)circuito (mantendo o número de vértices do (sub)circuito).

Com a união destas três operações (inserção, remoção e substituição) garante-se uma importante característica de uma estrutura de vizinhança que é ser possível partindo de qualquer solução obter outra qualquer solução por aplicação de uma sequência de operações.

Uma quarta operação, que pode ser incorporada nas heurísticas já referidas (ou noutras (meta-)heurísticas), é a operação de resequenciamento. Nesta operação é resolvido (por um método exacto ou heurístico) o PCV formado pelos vértices de um (sub)circuito o que, mantendo lucro porque os vértices do subconjunto são os mesmos, pode permitir uma redução da distância.

4.9.3 Programação inteira

Os modelos de programação inteira para os problemas do caixeiro viajante podem ser vistos como extensões aos modelos apresentados na secção anterior.

As variáveis de decisão

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

$$z_i = \begin{cases} 1, se\ vértice\ i\ é\ visitado \\ 0,\ caso\ contrário \end{cases}, \forall i \in N$$

As restrições de grau dos vértices passam a ser

$$\sum_{j:ij \in A} x_{ij} = z_i\,, \forall i \in N$$

$$\sum_{j:ij \in A} x_{ji} = z_i\,, \forall i \in N$$

$$z_1 = 1$$

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

$$z_i \in \{0,1\}, \forall i \in N$$

As restantes restrições e variáveis dos modelos discutidos anterioremente mantêm-se inalteráveis.

Para o PTP, a função objectivo é

$$Min\ z = \sum_{ij \in A} c_{ij} x_{ij} - \lambda \sum_{i \in N} p_i y_i$$

em que λ é uma constante.

Para o OP, a função objectivo é

$$Max\ z = \sum_{i \in N} p_i y_i$$

e adiciona-se a restrição

$$\sum_{ij \in A} c_{ij} x_{ij} \leq c_{max}$$

Para o PCP, a função objectivo é

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

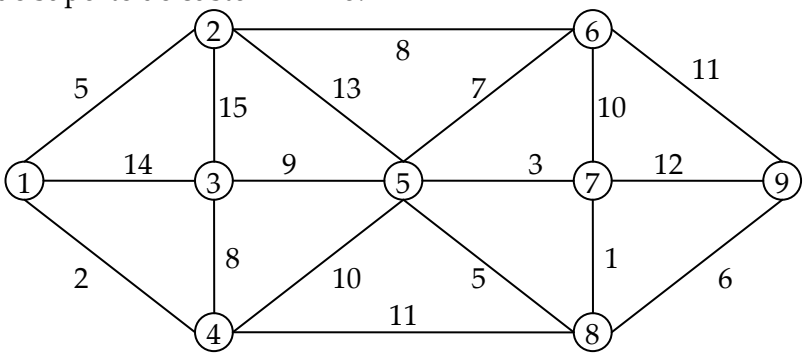
e adiciona-se a restrição

$$\sum_{i \in N} p_i y_i \geq p_{min}$$

4.10 Exercícios

Exercício 4.1 PCV simétrico: heurísticas construtivas

Para o PCV da figura, aplique as seguintes heurísticas:
a) Vizinho mais próximo partindo de cada um dos vértices;
b) Aresta com menor custo;
c) Inserção do vértice mais próximo partindo do subcircuito 1-2-3-4-1;
d) Clarke e Wright;
e) Árvore de suporte de custo mínimo.



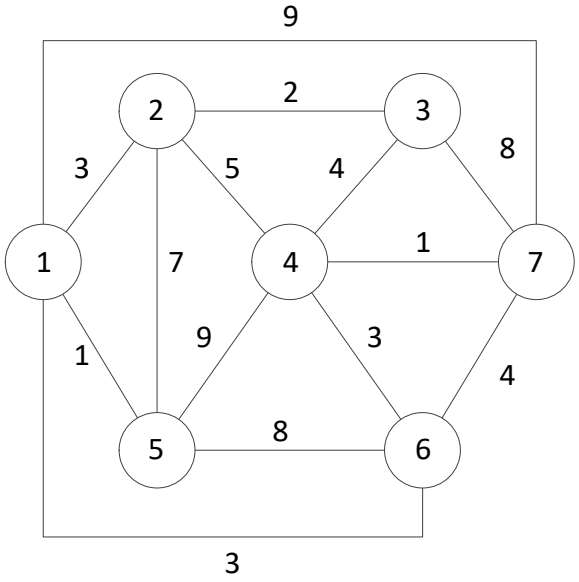
Exercício 4.2 PCV simétrico: heurísticas construtivas

1. Aplique as seguintes heurísticas ao PCV da figura.
a) Vizinho mais próximo partindo do vértice 1;
b) Aresta com menor custo;

2. A dado momento da aplicação da heurística de inserção do vértice mais próximo, o subcircuito actual é 3-4-7-3. Qual o vértice seguinte a inserir nesse subcircuito? Qual a variação de custo do actual para o novo subcircuito?

3. Aplique a heurística de Clarke e Wright partindo dos subcircuitos 1-2-4-5-1, 3-4-7-3 e 4-6-4.

4. A heurística da árvore de suporte de custo mínimo obtém uma solução admissível?



Exercício 4.3 PCV simétrico: heurísticas construtivas e pesquisa local

1. Para o PCV da tabela, aplique as seguintes heurísticas:
a) Vizinho mais próximo partindo do vértice 1;
b) Aresta com menor custo;
c) Inserção do vértice mais próximo partindo do subcircuito 1-2-3-4-1;
d) Clarke e Wright;
e) Árvore de suporte de custo mínimo.

2. Considerando o problema definido pelos 4 primeiros vértices e tendo como solução inicial a solução obtida com a heurística do vizinho mais próximo partindo de 1, obtenha uma solução óptima local para a vizinhança troca de dois vértices.

3, Considerando o problema definido pelos 5 primeiros vértices e tendo como solução inicial a solução obtida com a heurística do vizinho mais próximo partindo de 1, obtenha uma solução óptima local para a vizinhança troca de duas arestas (2-Opt).

	2	3	4	5	6
1	14	17	12	11	19
2	–	20	21	15	13
3	–	–	19	12	11
4	–	–	–	10	18
5	–	–	–	–	19

Exercício 4.4 PCV simétrico: heurísticas construtivas e pesquisa local

Considere a instância do PCV da tabela.

	1	*2*	*3*	*4*	*5*
1	-	3	4	1	9
2	3	-	4	5	3
3	4	4	-	8	2
4	1	5	8	-	9
5	9	3	2	9	-

1. Aplique as seguintes heurísticas:

a) Vizinho mais próximo partindo do vértice 1;

b) Aresta com menor custo;

c) Inserção do vértice mais próximo partindo do subcircuito 1-4-1;

d) Clarke e Wright;

e) Árvore de suporte de custo mínimo

2. Considerando a estrutura de vizinhança troca de duas arestas.

a) As soluções obtidas em 1a) e 1b) são vizinhas?

b) A solução obtida em 1a) é óptima local?

c) A solução obtida em 1b) é óptima local?

Exercício 4.5 PCV Euclidiano: heurísticas construtivas e pesquisa local

Considere um PCV Euclidiano com sete vértices com as coordenadas dadas na tabela.

i	1	2	3	4	5	6	7
x	0	0	1	2	2	3	5
y	0	3	1	0	1	0	5

1. Aplique as seguintes heurísticas:
a) Vizinho mais próximo partindo do vértice 1;
b) Aresta com menor custo;
c) Inserção do vértice mais próximo partindo do subcircuito 1-2-3-4-1;
d) Clarke e Wright;
e) Árvore de suporte de custo mínimo.

2. Mostre que a solução obtida com a heurística do vizinho mais próximo partindo de 1 não é uma solução óptima local para estrutura de vizinhança de troca de dois vértices.

3. Considere o problema com os cinco primeiros vértices da tabela. Através de pesquisa local com a vizinhança de troca de duas arestas (2-Opt), obtenha uma solução óptima local partindo da solução 1-5-4-3-2-1.

Exercício 4.6 PCV Euclidiano: heurísticas construtivas e pesquisa local

Considere um PCV Euclidiano com cinco vértices com as coordenadas dadas na tabela.

i	1	2	3	4	5
x	0	10	7	1	4
y	0	10	2	9	8

1. Aplique as seguintes heurísticas:
a) Vizinho mais próximo partindo do vértice 1;
b) Aresta com menor custo;
c) Inserção do vértice mais próximo partindo do subcircuito 1-2-3-4-1;
d) Clarke e Wright;
e) Árvore de suporte de custo mínimo.

2. Partindo da pior solução obtida com uma das heurísticas da alínea anterior, obtenha uma solução óptima local para as vizinhanças:
a) troca de dois vértices;
b) troca de duas arestas (2-Opt).

Exercício 4.7 PCV assimétrico: heurísticas construtivas

Aplique no problema do caixeiro viajante assimétrico com os custos dados na tabela as heurísticas:

	1	2	3	4	5	6
1	–	14	17	12	11	19
2	10	–	20	21	15	13
3	13	17	–	19	12	11
4	12	13	14	–	10	18
5	18	12	17	26	–	19
6	16	21	22	21	15	–

Aplique as seguintes heurísticas:
a) Vizinho mais próximo partindo de cada um dos vértices;
b) Aresta com menor custo;
c) Inserção do vértice mais próximo partindo do subcircuito 1-2-3-4-1;
d) Clarke e Wright.

Exercício 4.8 Aplicação a sequenciamento

Considere o problema de determinar a ordem pela qual um conjunto de seis tarefas (A, B, C, D, E e F) deve ser executado numa máquina. O tempo de execução de cada tarefa na máquina é independente da ordem pela qual as tarefas são realizadas. No entanto, o tempo de preparação da máquina para a realização de cada tarefa depende da tarefa que foi realizada imediatamente antes.

Na tabela seguinte são dados os tempos de execução e os tempos de preparação (em minutos) de cada tarefa. Por exemplo, a tarefa A demora 3 minutos a ser preparada se a tarefa anterior foi a B, a tarefa A demora 2 minutos a ser preparada se a tarefa anterior foi a C, e assim sucessivamente. Ainda por exemplo, a tarefa A demora 14 minutos a ser executada.

		Tarefa seguinte					
		A	B	C	D	E	F
Tarefa anterior	A	–	7	5	6	4	8
	B	3	–	4	6	7	8
	C	2	7	–	3	7	1
	D	3	2	8	–	2	9
	E	4	4	3	1	–	3
	F	7	4	2	2	4	–
	Tempo de execução	14	25	19	18	13	21

1. Mostre como cada solução deste problema de sequenciamento corresponde a uma solução de um problema do caixeiro viajante, identificando a que correspondem os nodos, os arcos e os custos dos arcos.

2. Obtenha uma solução com cada uma das seguintes heurísticas

a) Vizinho mais próximo partindo de cada um dos vértices;

b) Aresta de menor custo;

c) Inserção do vértice mais próximo partindo do subcircuito 1-2-3-4-1;

d) Clarke e Wright.

Exercício 4.9 *Programação inteira

Considere as restrições do modelo Dantzig-Fulkerson-Johnson

$$\sum_{j:ij\in A} x_{ij} = 1\,, \forall i \in N$$

$$\sum_{j:ji\in A} x_{ji} = 1\,, \forall i \in N$$

$$\sum_{ij\in A(S)} x_{ij} \leq |S| - 1, \forall S \subset N$$

a) Indique as restrições que não estão a ser respeitadas na solução $x_{12} = x_{21} = x_{34} = x_{45} = x_{53} = 1$ e todas as restantes variáveis iguais a zero.

b) Indique uma solução que satisfaça os dois primeiros grupos de restrições mas não satisfaça pelo menos uma das restrições do terceiro grupo.

c) Indique o valor das variáveis de decisão (x e u) do modelo de Miller-Tucker-Zemlin na solução 13-34-45-52-21 de um PCV com cinco nodos.

Exercício 4.10 *Programação inteira

Considere o PCV da tabela.

	1	2	3	4	5
1	–	4	7	2	1
2	3	–	3	1	12
3	4	5	–	2	6
4	5	1	6	–	10
5	6	9	2	5	–

1. Apresente os seguintes modelos:
a) Dantzig-Fulkerson-Johnson;
b) Miller-Tucker-Zemlin;
c) fluxos (agregado);
d) fluxos (desagregado);
e) etapas.

2. Utilizando um *solver* de programação inteira, obtenha uma solução óptima com base em cada um dos modelos da alínea anterior.

Exercício 4.11 *GRASP

Aplique a meta-heurística GRASP no PCV definido pela tabela para valores de α de 0.5 e 1. Na componente aleatória do algoritmo, tome decisões arbitrárias. Utilize a heurística construtiva do vizinho mais próximo, seleccionando, em cada iteração, arbitrariamente o vértice inicial, e utilize a heurística de troca de duas arestas para a pesquisa local. Use como critério de paragem não ter havido melhoria da solução incumbente.

	2	3	4	5
1	4	7	2	1
2	–	3	1	12
3	–	–	2	6
4	–	–	–	10

4.11 Bibliografia

- J. E. Beasley , OR-Library, http://people.brunel.ac.uk/~mastjjb/jeb/info.html
- D. Feillet, P. Dejax, M. Gendreau. "Traveling salesman problems with profits." Transportation science 39.2 (2005): 188-205.
- D. S. Johnson, L. A. McGeoch. "The traveling salesman problem: A case study in local optimization." Local search in combinatorial optimization 1 (1997): 215-310.
- S. Mertens, TSP Algorithms in Action Animated Examples of Heuristic Algorithms, http://wase.urz.uni-magdeburg.de/mertens/TSP/
- J. F. Oliveira, M. A. Carravilla, Heuristics and Local Search, http://paginas.fe.up.pt/~mac/ensino/docs/OR/HowToSolveIt/ConstructiveHeuristicsForTheTSP.pdf, 2001

• A. J. Orman, H.P. Williams, A Survey of Different Integer Programming Formulations of the Travelling Salesman Problem, Optimisation, Econometric and Financial Analysis, Advances in Computational Management Science 9, pp 91-104, 2007

• G. Reinelt, The Traveling Salesman Computational Solutions for TSP Applications, Lecture Notes in Computer Science 840, online edition, 2001.

• TSP - Principal sítio sobre o PCV que inclui muita informação, muitas aplicações e permite download do software concorde http://www.math.uwaterloo.ca/tsp/index.html

4.12 Resultados de aprendizagem

• Aplicar heurísticas construtivas para resolver instâncias de pequena dimensão do problema do caixeiro viajante.

• Aplicar heurísticas de pesquisa local para resolver instâncias de pequena dimensão do problema do caixeiro viajante.

• *Formular o problema do caixeiro viajante (e variantes com lucros) através de modelos de programação inteira.

• *Resolver problema do caixeiro viajante (e variantes com lucros) com software, nomeadamente o Solver e openSolver para Excel e o IBM ILOG CPLEX Optimization Studio.

5 Fluxos

5.1 Introdução

Os modelos de fluxos abordados neste capítulo permitem obter soluções para problemas em que se pretende que quantidades de uma entidade (ou comodidade / "commodity") sejam enviados entre locais numa rede existindo um objectivo conhecido.

Estes problemas foram estudados pela primeira vez por Kantorovich na União Soviética no final dos anos 1930 e posteriormente, em meados dos anos 1950, por Ford e Fulkerson (como referido por Schrijver (2002)).

5.2 Fluxo de custo mínimo

5.2.1 Definição e notação

No problema do fluxo de custo mínimo pretende-se enviar um produto dos nodos onde está disponível (origens) em quantidades conhecidas (oferta) para os nodos onde é requerida (destinos) em quantidades conhecidas (procura). Associado a cada arco, existe um custo unitário e, possivelmente, uma capacidade. Pretende-se minimizar o custo total do envio das origens para os destinos das unidades requeridas por estes últimos.

Numa perspectiva de redes, o problema do fluxo de custo mínimo consiste em, dada uma rede em que os nodos têm ofertas e procuras e os arcos têm capacidades e custos unitários, determinar os fluxos em cada arco de tal forma que as ofertas e procuras são satisfeitas, as capacidades não são excedidas e o custo total é o menor possível.

Exemplo 5.1 Fluxo de custo mínimo

Uma empresa de distribuição tem um produto armazenado em duas localizações distintas. Na localização 1 tem 50 unidades e na localização 4 tem 10 unidades. Dois clientes, localizados em 2 e 5, pretendem adquirir 40 e 20 unidades do produto em causa.

Na Figura 5.1 representam-se, numa rede, todas as localizações relevantes deste problema, bem como o custo associado a transportar uma unidade do produto entre aquelas em que é possível fazê-lo. Não é possível transportar mais de 40 unidades entre cada par de localizações.

O problema do fluxo de custo mínimo consiste em determinar a forma mais económica de transportar as 60 unidades com o menor custo possível.

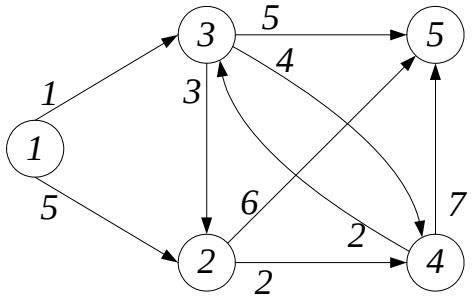


Figura 5.1. Exemplo de problema de fluxo de custo mínimo.

■

A rede é constituída por n nodos e m arcos. O conjunto dos índices dos nodos representa-se por N. Cada arco é identificado pelos índices dos dois nodos que liga, ij. O conjunto dos arcos representa-se por A. A capacidade de um arco ij (associada, por exemplo, à largura de banda numa rede de comunicações) é representada por u_{ij}. O custo de uma unidade atravessar um arco (por exemplo, o atraso de uma mensagem ao ser transmitida por uma linha que une dois computadores) é representado por c_{ij} (assumindo-se que $c_{ij} \geq 0, \forall ij \in A$). A oferta (por exemplo, número de unidades de um produto existentes num armazém) e a procura de um nodo (por exemplo, número de unidades de um produto que têm ser entregues a um cliente) são representadas por b_i. Se $b_i > 0$, o nodo é uma origem (com oferta de b_i unidades). Se $b_i < 0$, o nodo é um destino (com procura de $-b_i$ unidades). Se $b_i = 0$, o nodo não é origem nem destino, sendo nesse caso um nodo de passagem (ou transbordo).

5.2.2 Programação linear

Um modelo de programação linear para o problema do fluxo de custo mínimo do Exemplo 5.1 baseia-se nas seguintes variáveis de decisão

$$x_{ij} - quantidade\ a\ transportar\ do\ local\ i\ para\ o\ local\ j, \forall ij \in A.$$

$$Min = 5x_{12} + x_{13} + 2x_{24} + 6x_{25} + 3x_{32} + 4x_{34} + 5x_{35} + 2x_{43} + 7x_{45}$$

$$sujeito\ a;$$

$$x_{12} + x_{13} = 50$$

$$x_{24} + x_{25} + 40 = x_{12} + x_{32}$$

$$x_{32} + x_{34} = x_{13} + x_{43}$$

$$x_{43} + x_{45} = x_{24} + x_{34} + 10$$

$$20 = x_{25} + x_{35} + x_{45}$$

$$0 \leq x_{ij} \leq 40, \forall ij \in \{(1,2), (1,3), (2,4), (2,5), (3,2), (3,4), (3,5), (4,3), (4,5)\}$$

A última expressão pode ser substituída por

$$0 \leq x_{ij} \leq 40, \forall ij \in A$$

assumindo que é claro o conjunto dos arcos a considerar. Neste caso A encontra-se definido na representação do grafo, sendo claro que $A = \{(1,2), (1,3), (2,4), (2,5), (3,2), (3,4), (3,5), (4,3), (4,5)\}$.

■

Um modelo de programação linear para o problema do fluxo de custo mínimo geral baseia-se nas seguintes variáveis de decisão

x_{ij} – fluxo no arco ij, $\forall ij \in A$

e é

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

$$sujeito\ a:$$

$$\sum_{j:ij \in A} x_{ij} - \sum_{j:ji \in A} x_{ji} = b_i, \forall i \in N$$

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

É de salientar que caso as ofertas, procuras e capacidades tenha valores inteiros, existe uma solução óptima que é inteira, não havendo portanto a necessidade de incluir as restrições de integralidade das variáveis mesmo quando a aplicação o exige.

5.2.3 *Simplex de rede

Dado a sua estrutura, existe uma especialização do algoritmo simplex (usado para obter a solução óptima de problemas de programação linear), designada por simplex de rede, que é muito mais eficiente na sua resolução do que a utilização do algoritmo simplex geral. Descreve-se aqui a versão do algoritmo simplex de rede para problemas de fluxo de custo mínimo sem capacidades com base no problema do Exemplo 5.1 (assim, não são tidas em conta as restrições de capacidade de 40 unidades).

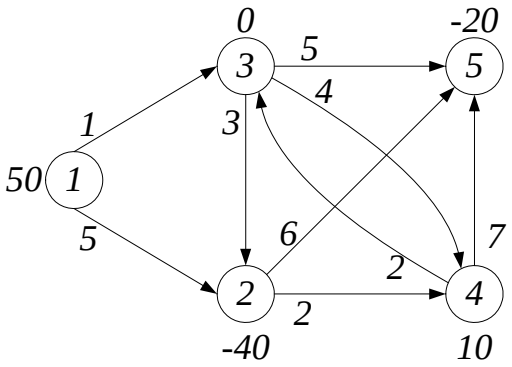


Figura 5.2. Problema de fluxo de custo mínimo.

Um algoritmo simplex baseia-se na propriedade de para qualquer problema de programação linear que tenha pelo menos uma solução óptima finita (não é impossível nem com solução ilimitada), existe uma óptima que resulta de anular $n - m$ variáveis de decisão e resolver o sistema de m equações e m incógnitas (as variáveis de decisão que não foram anuladas). Uma solução que é obtida desta forma (anular $n - m$ variáveis de decisão e resolver o sistema de m equações e m incógnitas) designa-se por solução básica. No caso dos valores das variáveis de decisão obtidos pela resolução do sistema de equações serem todos não negativos, a solução é uma solução básica admissível. Se houver, pelo menos, uma variável de decisão negativa a solução é uma solução básica não admissível.

Considere-se o exemplo que tem nove variáveis de decisão e cinco restrições. De facto, uma das restrições é redundante[16], o que se intui por o fluxo em todos os arcos ficar determinado por aplicação da conservação de fluxo em apenas quatro nodos (usando o facto da oferta total ser igual à procura total).

Assim, consideramos nove variáveis de decisão e quatro equações. Tal significa que uma solução básica consistirá em ter cinco arcos sem fluxo e quatro arcos com fluxo positivo[17]. Notando que apenas estes quatro arcos terão de permitir o transporte das quantidades de qualquer origem para qualquer destino, chegamos à conclusão que estes arcos terão de corresponder a uma estrutura em que existe um caminho entre cada par de nodos. Essa estrutura é a árvore de suporte (conjunto de $n - 1$ arestas - em que n é o número de vértices da rede - que não forma (sub)circuitos).

Assim, uma solução básica para o problema de fluxo de custo mínimo está associada a uma *árvore de suporte*. Para problemas de fluxo de custo mínimo interessa definir árvores de suporte admissíveis (é possível transportar todas as unidades das origens para os destinos utilizando apenas os arcos da árvore de suporte, ver exemplo da Figura 5.3) e não admissíveis (não é possível transportar todas as unidades das origens para os destinos utilizando apenas os arcos da árvore de suporte, ver exemplo da Figura 5.4).

A cada árvore de suporte corresponde uma e uma só solução que se obtém fixando os arcos que não fazem parte da árvore a 0 e resolvendo as equações de conservação de fluxo (a solução pode não ser admissível por ter fluxos negativos).

---

[16] Mais correctamente, existem quatro restrições linearmente independentes, uma é linearmente dependente.

[17] Eventualmente, algum(ns) deste(s) quatro arcos básicos poderá(ão) ter fluxo zero.

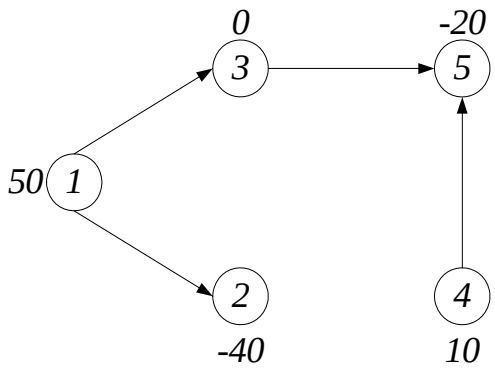


Figura 5.3. Árvore de suporte admissível.

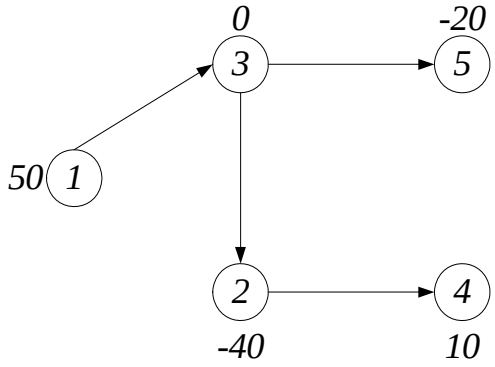


Figura 5.4. Árvore de suporte não admissível. Para ser possível transportar o fluxo a partir de 4, seria necessário que o fluxo no arco 24 fosse negativo o que não é admissível.

Os passos do algoritmo simplex de rede são semelhantes aos passos do algoritmo simplex primal, mas as operações de mudança de base são feitas de forma muito mais eficiente.

O algoritmo simplex de rede parte de uma árvore de suporte correspondente a uma solução básica admissível. De seguida, para cada arco que não faz parte da árvore (arco não básico), determina a variação do custo se o arco passar a fazer parte da árvore. Se a variação de custo para cada um dos arcos for maior ou igual a zero, a solução actual é óptima. Caso contrário, adiciona-se o arco com menor variação de custo à árvore de suporte e remove-se um arco de forma a manter-se uma árvore de suporte (apenas existe um arco nestas condições).

O cálculo das variações de custo pode ser feito usando a teoria da dualidade. Por essa razão, apresenta-se o modelo dual do modelo de fluxo de custo mínimo, em que y_i é a variável dual associada ao nodo i:

$$Max\ w = \sum_{i=1}^{n} b_i y_i$$

$$sujeito\ a:$$

$$y_i + y_j \leq c_{ij}, \forall ij \in A$$

$$y_i\ livre, i = 1, \ldots, n$$

As condições de optimalidade da programação linear traduzem-se no problema de fluxo de custo mínimo em uma solução x ser óptima se:

(i) x é admissível

(ii) se $x_{ij} > 0$ então $y_i + y_j = c_{ij}$

(iii) se $x_{ij} = 0$ então $y_i + y_j \leq c_{ij}$

A primeira condição garante a admissibilidade primal, as duas outras condições garantem o respeito do teorema da folga complementar e a admissibilidade dual. Por definição de custo reduzido, a variação do custo se uma variável não básica x_{ij} entrar para base é a folga da condição *(iii)*, ou seja $c_{ij} - y_i - y_j$.

Exemplificando, uma solução básica admissível é:

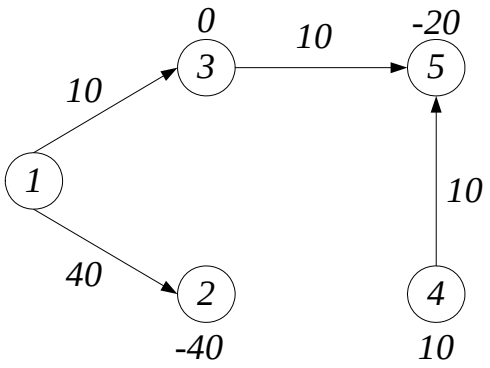


Figura 5.5. Solução primal.

O valor das variáveis duais é obtido com base na resolução do sistema de equações da Tabela 5.1.

Arco básico	Equação
(1,2)	$y_1 - y_2 = 5$
(1,3)	$y_1 - y_3 = 1$
(3,5)	$y_3 - y_5 = 5$
(4,5)	$y_4 - y_5 = 7$

Tabela 5.1. Equações para cálculo das variáveis duais no algoritmo simplex de rede.

Uma solução dual é $y_1 = 0, y_2 = -5, y_3 = -1, y_4 = 1$ e $y_5 = -6$.

Os custos reduzidos dos arcos não básicos são:

$$\overline{c_{24}} = c_{24} - y_2 + y_4 = 2 - (-5) + (1) = 8$$
$$\overline{c_{25}} = c_{25} - y_2 + y_5 = 6 - (-5) + (-6) = 5$$
$$\overline{c_{32}} = c_{32} - y_3 + y_2 = 3 - (-1) + (-5) = -1$$
$$\overline{c_{34}} = c_{34} - y_3 + y_4 = 4 - (-1) + (1) = 6$$
$$\overline{c_{43}} = c_{43} - y_4 + y_3 = 2 - (1) + (-1) = 0$$

A solução não é óptima porque o custo reduzido do arco (3,2) é negativo. O arco (3,2) entra para a base. Deixa-se de ter uma árvore de suporte (tem-se n arcos logo existe um ciclo). Forma-se sempre um e um único ciclo ao adicionar-se um arco a uma árvore de suporte.

Um arco tem de sair da base. Se se aumentar o fluxo em (3,2) de d, para se continuar a respeitar as restrições de conservação de fluxo, o fluxo em (1,2) tem de

diminuir d e o fluxo em (1,3) tem de aumentar d. Para aumentar o mais possível o valor do fluxo em (3,2) e manter a validade da base, $d = 40$ e o arco (1,2) sai da base, como representado na figura seguinte.

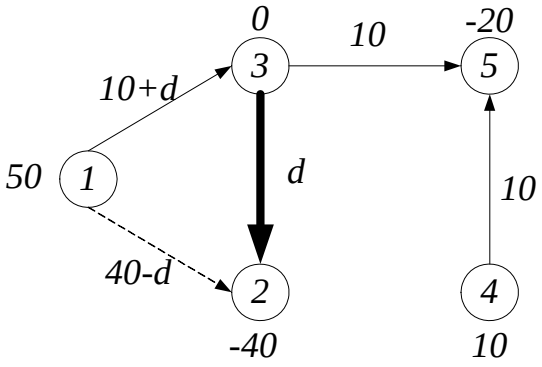


Figura 5.6. Mudança de base.

O custo da solução anterior era de 330, o custo da solução actual, representada na figura seguinte, é 330 – 1.40 = 290.

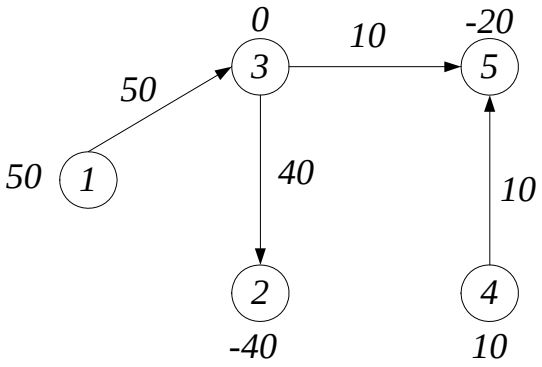


Figura 5.7. Nova solução primal.

Calculando as duais e os custos reduzidos conclui-se que esta solução é óptima.

5.2.4 Fluxo de custo mínimo em folha de cálculo

Dada a sua estrutura, é extremamente simples obter a solução óptima de um problema de transportes numa folha de cálculo (com um *solver* de programação linear).

Na Figura 5.8 apresenta-se um modelo numa folha de cálculo (com um *solver* de programação linear) para o problema de fluxo de custo mínimo relativo ao Exemplo 5.1.

As variáveis de decisão correspondem às células B3:J3. As restrições são L2:L10=N6:N10 e B3:J3<=B1:J1. A função objectivo corresponde à célula L12. Nas células L6 a L10 e L12 têm fórmulas semelhantes. Por exemplo, na célula L8 tem-se "=SUMPRODUCT($B$3:$J$3,B8:J8)" e na célula L12 "=SUMPRODUCT($B$3:$J$3,B12:J12)".

A matriz B6:J10 define a rede, o que permite uma definição expedita do membro esquerdo das restrições de conservação de fluxo através fórmulas acima referidas.

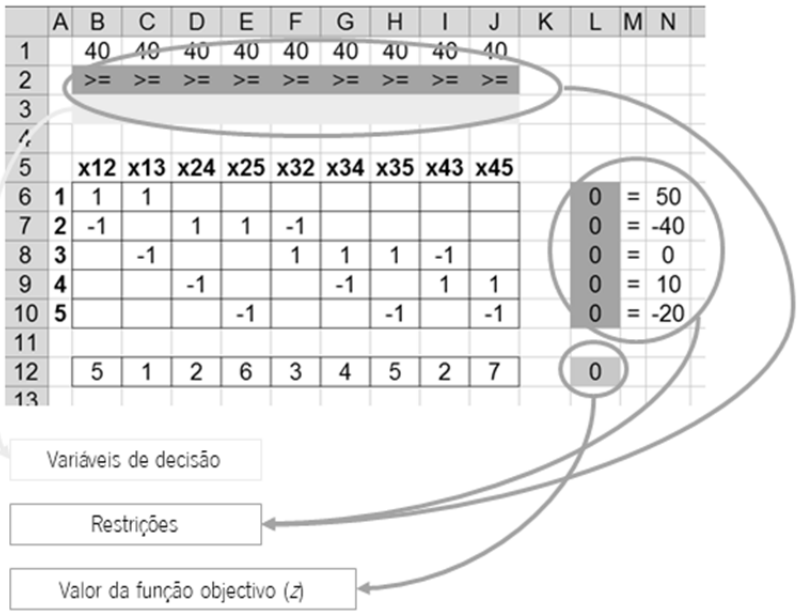


Figura 5.8. Fluxo de custo mínimo numa folha de cálculo.

Após inserir as referências nas janelas do *solver* e optimizado, deve-se ter o cuidado de verificar que a mensagem retornada pelo *solver* indica que foi obtida uma solução óptima.

5.3 Problema de transportes

Num problema de transportes pretende-se determinar de que forma transportar unidades de um determinado produto entre origens e destinos de forma a minimizar o custo total. Numa perspectiva de redes, o problema de transportes consiste em, dada uma rede em que cada nodo tem uma oferta ou uma procura e os arcos têm custos unitários, determinar os fluxos em cada arco de tal forma que as ofertas e procuras são satisfeitas e o custo total é o menor possível.

O problema de transportes é um caso particular do problema de fluxo de custo mínimo em que cada arco liga uma origem (nodo com oferta) a um destino (nodo com procura), sendo portanto a rede subjacente ao problema uma rede bipartida.

Exemplo 5.2 Transportes

Uma empresa tem fábricas em Aveiro, Braga e Coimbra que produzem no máximo 75, 125 e 150 unidades de um produto por semana. A empresa entrega, todas as semanas, 50, 100 e 200 unidades em Viseu, Guarda e Lisboa, respectivamente.

Assumindo que o custo de transporte de cada unidade é proporcional à distância entre as cidades (dada na tabela), qual o melhor plano de transporte semanal?

Distância (Km)	Viseu	Guarda	Lisboa
Aveiro	95	180	244
Braga	186	263	366
Coimbra	96	168	196

■

Em relação ao problema do fluxo de custo mínimo, por clareza do modelo, altera-se a notação relativa às ofertas e procuras da seguinte forma: a_i é a oferta da origem i, b_j é a procura do destino j. Além disso, considera-se que n representa o número de origens e m o número de destinos.

No exemplo, as variáveis de decisão são

x_{ij} – quantidade transportada entre a cidade i e a cidade j, $i = 1,2,3;\ j = 1,2,3$

e o modelo de programação linear é

$$Min\ z = 95x_{11} + 180x_{12} + 244x_{13} + 186_{x21} + 263x_{22} + 366x_{23} + 96x_{31} + 168x_{32} + 196x_{33}$$

$sujeito\ a$:

$$x_{11} + x_{12} + x_{13} \leq 75$$
$$x_{21} + x_{22} + x_{23} \leq 125$$
$$x_{31} + x_{32} + x_{33} \leq 150$$
$$x_{11} + x_{21} + x_{31} = 50$$
$$x_{12} + x_{22} + x_{32} = 100$$
$$x_{13} + x_{23} + x_{33} = 200$$
$$x_{11}, x_{12}, \dots, x_{33} \geq 0$$

As variáveis de decisão para o problema geral são

x_{ij} – quantidade transportada entre o local i e o local j (ou fluxo no arco ij) entre a cidade *i* e a cidade *j, i=1,2,3; j=1,2,3*

e modelo geral é

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

$sujeito\ a$:

$$\sum_{ij \in A} x_{ij} \leq a_i, i = 1, .., n$$

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

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

Este modelo pressupõe que a quantidade total oferecida pelas origens, $\sum_i a_i$, é maior ou igual que a quantidade procurada pelos destinos, $\sum_j b_j$. No caso de tal não se verificar o problema passa a ser impossível ou deverá ser considerado que se pretende entregar todas as unidades existentes nas origens com o menor custo possível, sabendo-

se de antemão que a procura não pode ser toda satisfeita (as restrições das origens passam a igualdade e as dos destinos a restrições de menor ou igual). Se a oferta total for igual à procura total, $\sum_i a_i = \sum_j b_j$, as restrições podem ser transformadas em igualdade (embora não seja necessário).

Tal como no problema do fluxo de custo mínimo, é de salientar que, no problema de transportes, caso as ofertas e as procuras tenha valores inteiros, existe uma solução óptima que é inteira, não havendo portanto a necessidade de incluir as restrições de integralidade das variáveis mesmo quando a aplicação o exige.

5.3.2 Simplex para transportes e árvores de suporte

Sendo um caso particular do problema do fluxo de custo mínimo, existe sempre uma solução óptima para o problema de transportes que corresponde a uma árvore de suporte (ignorando a orientação dos arcos) tal que todos os arcos com fluxo fazem parte da árvore de suporte.

A cada árvore de suporte corresponde uma e uma só solução que se obtém fixando os arcos que não fazem parte da árvore a 0 e resolvendo as equações de conservação de fluxo (a solução pode não ser admissível por ter fluxos negativos).

Por exemplo, a solução da rede à direita na Figura 5.9 para o problema definido na rede da esquerda corresponde a uma árvore de suporte (em que implica fluxos negativos, logo a solução é não admissível).

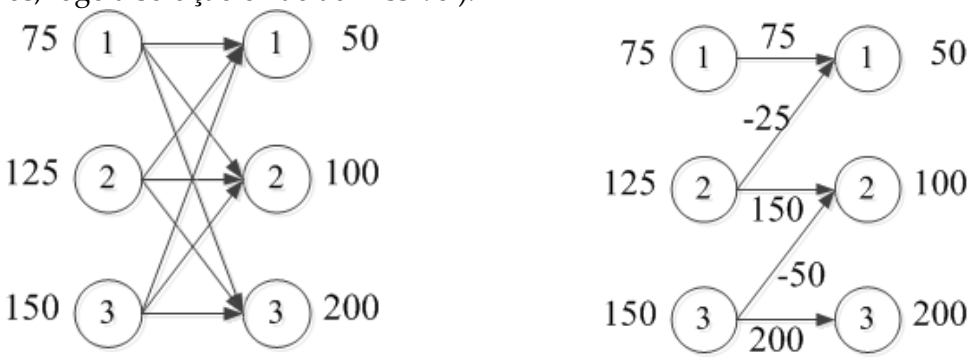


Figura 5.9. Exemplo de solução que corresponde a árvore de suporte mas não é admissível.

Asssume-se que a oferta total é igual procura total. Caso tal não se verifique no problema original deverá ser adicionada um destino (origem) fictício que absorva o excesso de oferta (procura) relativamente à procura (oferta).

Um algoritmo para o problema de transportes é:

```
1. Obter uma solução admissível que corresponda a uma árvore de
        suporte
2. Para todos arcos que não fazem parte da árvore, calcular a
        variação de custo se o arco passar a fazer parte da árvore
        e transportar uma unidade, mantendo a admissibilidade da
        solução
3. Se não há variações de custo negativas, parar, a solução é
        óptima
4. Escolher o arco que com uma variação de custo mais negativa
5. Actualizar a árvore de suporte de forma a que a solução
        permaneça admissível e ir para 2
```

No passo 2, ao adicionar-se um arco à árvore, forma-se um e um só ciclo que permite o cálculo da variação do custo ou, em alternativa, a variação de custo pode ser calculada com base na teoria da dualidade (ver secção seguinte). Esta variação de custo designa-se por custo reduzido.

No passo 5, o arco que sai da árvore é que o primeiro se anula com o aumento do fluxo do arco que entra na árvore.

Exemplifica-se agora aplicação do algoritmo com base no Exemplo 5.2.

No passo 1, obtém-se uma solução admissível que corresponda a uma árvore de suporte. Tal pode ser feito através do método dos custos mínimos. Neste método, sequencialmente (por ordem crescente dos custos unitários) atribui-se o maior valor possível a cada transporte, actualizando as quantidades disponíveis e requeridas. No exemplo, o menor custo é o da célula Aveiro-Viseu, logo a solução (incompleta) passa a ser

	Viseu	Guarda	Lisboa			
Aveiro	**50**			50	=	75
Braga	0				=	125
Coimbra	0				=	150
	50					
	=	=	=			
	50	100	200			

A célula com menor custo das que ainda podem ter valores positivos é a Coimbra-Guarda (note-se que Viseu já tem a procura satisfeita logo Braga-Viseu e Coimbra-Viseu têm valor zero na solução que está a ser construída).

	Viseu	Guarda	Lisboa			
Aveiro	50	0		50	=	75
Braga	0	0			=	125
Coimbra	0	100		100	=	150
	50	100				
	=	=	=			
	50	100	200			

Com o mesmo raciocínio obtém-se a solução

	Viseu	Guarda	Lisboa			
Aveiro	50	0	25	75	=	75
Braga	0	0	125	125	=	125
Coimbra	0	100	50	100	=	150
	50	100	200			
	=	=	=			
	50	100	200			

Esta solução tem custo total de 83200.

Representando a solução numa rede (esquerda da Figura 5.10) é clara a árvore de suporte (figura da direita) que lhe está associada.

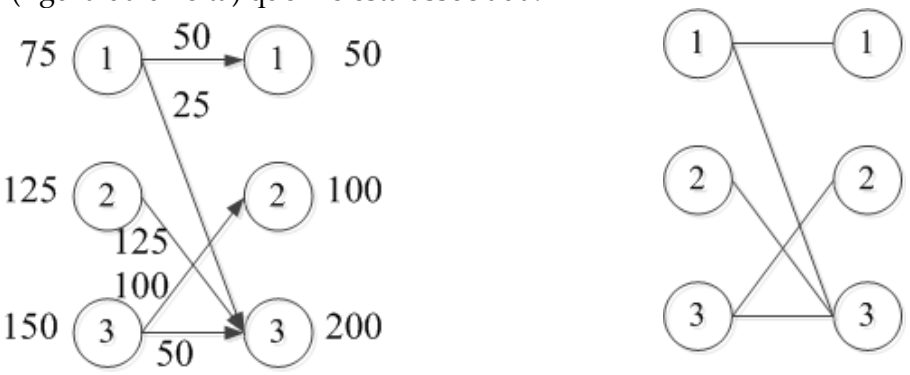


Figura 5.10. Solução inicial.

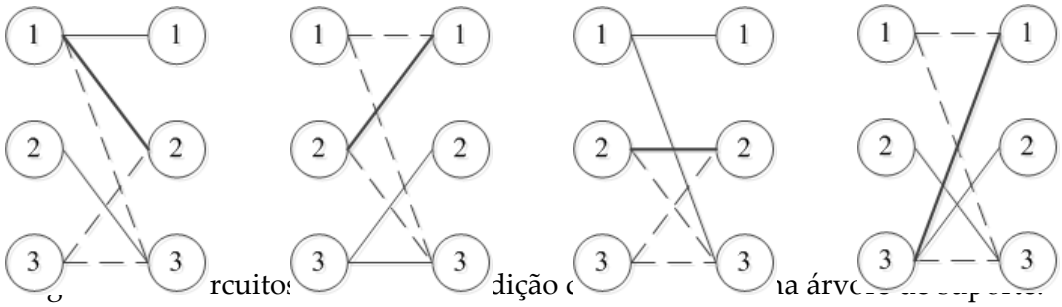
Se no final da aplicação do método dos custos mínimos, o número de arcos fosse inferior a $n+m-1$, seria necessário acrescentar arcos (com quantidades transportadas de 0) garantindo a existência de uma árvore de suporte associada à solução.

No segundo passo, para todos arcos que não fazem parte da árvore de suporte, calcula-se a variação de custo se o arco passar a fazer parte da árvore e transportar uma unidade, mantendo a admissibilidade da solução (conservação de fluxo nos nodos).

Aumentando uma unidade de fluxo num arco existe sempre um e só um ciclo que permite que as ofertas e procuras permaneçam as mesmas.

Por exemplo, para o arco 12, a variação de custo por aumento de uma unidade de fluxo é +180 (arco 12) – 168 (arco 32) + 196 (arco 33) – 244 (arco 13) = −36.

Na Figura 5.11 apresentam-se os circuitos que se formam para cada um dos arcos não básicos, a partir dos quais se calculam as variações de custo (custos reduzidos) que são dados na Tabela 5.2.



rcuitos dição a árvore de suporte.

	Viseu	Guarda	Lisboa
Aveiro		–36	
Braga	–31	–75	
Coimbra	+49		

Tabela 5.2. Custos reduzidos associados à primeira solução.

Como há custos reduzidos negativos, a solução não é óptima. Escolhe-se portanto o arco que com um custo reduzido mais negativo que é o arco 22.

É então necessário determinar um arco para sair da base, mantendo assim a estrutura da árvore e assegurando que a solução permanece admissível. Tal é feito tendo em conta que quanto maior a quantidade transportada entre 2 e 2, mais o custo diminui, no entanto, mas não pode haver fluxos negativos. A maior quantidade que o arco 22 é determinada pelos arcos do circuito cujo fluxo tem de diminuir para o fluxo de 22 aumentar, ou seja, os arcos 32 e 13. Assim o fluxo em 22 pode aumentar até 100 (um valor maior implicaria um fluxo negativo em 32 – um fluxo maior que 125 implicaria um fluxo negativo também em 13). O arco 32 sai da árvore de suporte (porque é o primeiro cujo fluxo se anula com o aumento do fluxo no arco 22). Quando vários arcos ficam com fluxo zero quando um arco é adicionado à árvore de suporte, apenas um deverá ser removido, mantendo-se assim uma árvore de suporte.

A segunda solução e árvore de suporte correspondente são dadas na Figura 5.12.

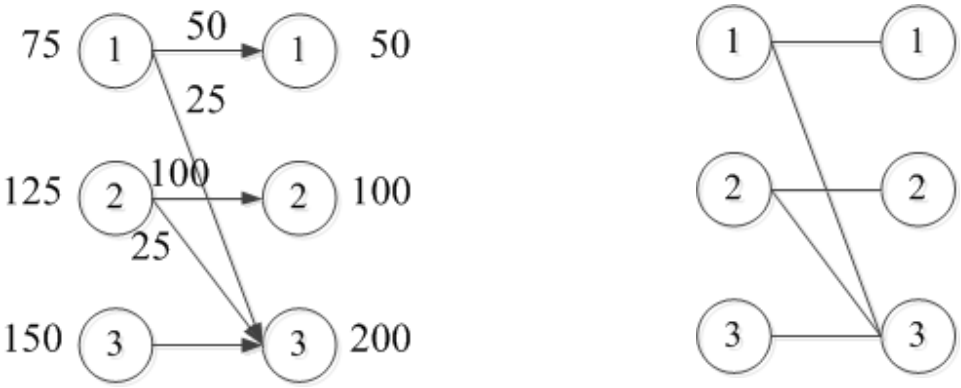


Figura 5.12. Segunda solução.

Esta solução tem custo = 83200 – 100.75 = 75700.

Calculando os custos reduzidos, obtém-se os valores dados na Tabela 5.3.

	Viseu	Guarda	Lisboa
Aveiro		39	
Braga	–31		
Coimbra	49	75	

Tabela 5.3. Custos reduzidos da segunda solução.

Como existe um arco com custo reduzido negativo, a solução não é óptima. O arco 21 entra para a base e sai o arco 23.

A terceira solução é dada na Figura 5.13. Terceira solução.Figura 5.13. O custo da solução é 75700 – 31.25 =74925.

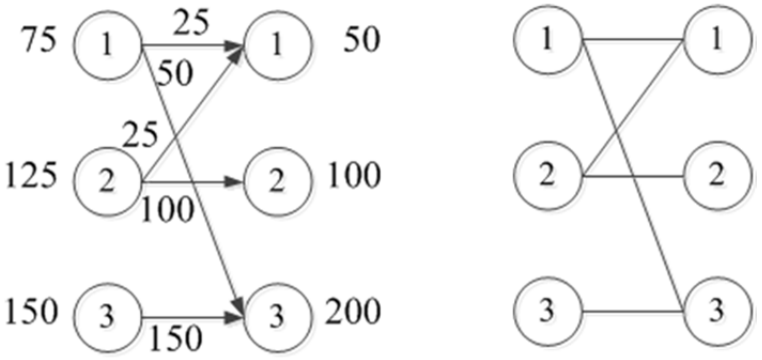


Figura 5.13. Terceira solução.

Os custos reduzidos associados a esta solução são dados na Tabela 5.4.

	Viseu	Guarda	Lisboa
Aveiro		8	
Braga			31
Coimbra	49	44	

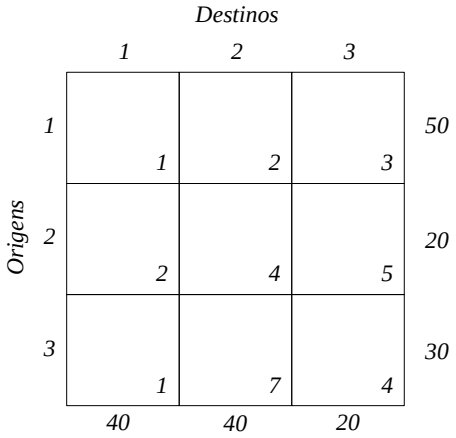
Tabela 5.4. Custos reduzidos da terceira solução.

Como não existem arcos com custos reduzidos negativos, a solução actual é uma solução óptima.

5.3.3 Simplex para transportes e dualidade

Quadro de transportes

Dada a sua estrutura, o modelo de transportes pode ser representado num quadro em que cada variável de decisão está associada a uma célula e cada linha e cada coluna estão associadas a uma restrição (representando-se também o termo independente). A função objectivo é representada através do coeficiente de cada variável de decisão no canto inferior direito da célula respectiva.



Para aplicar o algoritmo simplex para transportes, a oferta total tem de ser igual à procura total (o quadro tem de estar balanceado). Quando há oferta em excesso, deve ser introduzido um destino fictício (coluna) que a absorva. Quando há procura em excesso, deve ser introduzida uma origem fictícia (linha) que a forneça. O quadro seguinte é um exemplo para um caso em que a oferta total era de 110 e a procura total de 140.

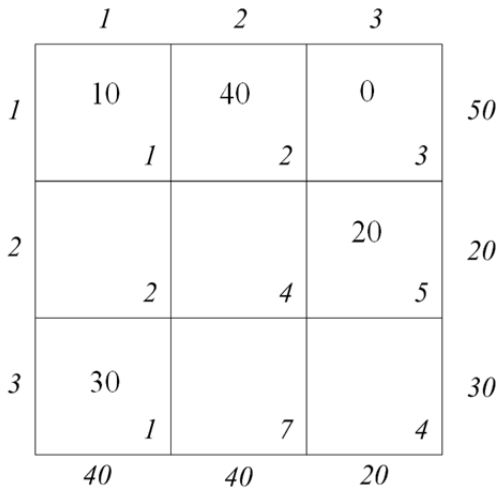
x_{11} 1	x_{12} 2	x_{13} 3	50
x_{21} 2	x_{22} 4	x_{23} 5	25
x_{31} 1	x_{32} 7	x_{33} 4	35
s_1 0	s_2 0	s_3 0	30
70	45	25	

Frequentemente os custos de transporte a partir da origem fictícia são zero (correspondem a transportes não efectuados na realidade). Em algumas situações podem não ser. Por exemplo, num problema de produção, se a capacidade de produção for de 110 unidades e se se tiver encomendas de 140 unidades, a origem fictícia pode estar associada a um fornecedor externo ou a trabalho em horas extraordinárias, tendo os custos unitários valores adequados a essas situações.

Solução básica inicial

Considerando que, depois do quadro balanceado, o problema de transportes tem n origens e m destinos (logo tem $n+m$ restrições) uma base tem $n+m-1$ variáveis (uma restrição é redundante por causa da oferta total ser igual à procura total). Tal significa que uma solução óptima tem no máximo $n+m-1$ variáveis com valor positivo. O algoritmo simplex para transportes começa com uma solução básica admissível, testa a sua optimalidade e, caso a solução não seja óptima, passa para outra solução básica admissível com menor custo. A diferença fundamental em relação ao algoritmo simplex é que a passagem de uma solução básica para outra é feita de forma muito mais eficiente, já que não envolve a resolução de sistemas de equações, mas apenas operações de soma e subtracção.

Um método para obter uma solução básica admissível é designado por método dos custos mínimos porque atribui a maior quantidade possível às células partindo das que têm menor custo. No exemplo, $x_{31}=Min\{30,40\}=30$ (também se podia ter começado por x_{11}). A procura de 1 passa a ser *10*. A oferta de 3 passa a ser *0*, logo $x_{32}=x_{33}=0$. $x_{11}=Min\{50,10\}=10$. A oferta de 1 passa a ser *40*. A procura de 1 passa a ser *0*, logo $x_{21}=0$. Continuando com o mesmo procedimento obtém-se a solução representada a seguir.



Note-se que a solução obtida é uma solução degenerada (tem uma variável básica com valor zero). A variável x_{13} foi seleccionada arbitrariamente para entrar na base de entre todas as não básicas, ficando a base com o número certo de variáveis $n + m - 1$, neste exemplo, $3 + 3 - 1$.

Problema dual e condições de optimalidade

O problema dual do problema de transportes é

$$Max\ w = \sum_{i=1}^{n} a_i u_i + \sum_{j=1}^{m} b_j v_j$$

$$sujeito\ a:$$

$$u_i + v_j \leq c_{ij}, \forall ij \in A$$

$$u_i\ livre, i = 1, \dots, n$$

$$v_j livre, j = 1, \dots, m$$

As condições de optimalidade da programação linear traduzem-se no problema de transportes em uma solução x ser óptima se:

(i) x é admissível

(ii) se $x_{ij} > 0$ então $u_i + v_j = c_{ij}$

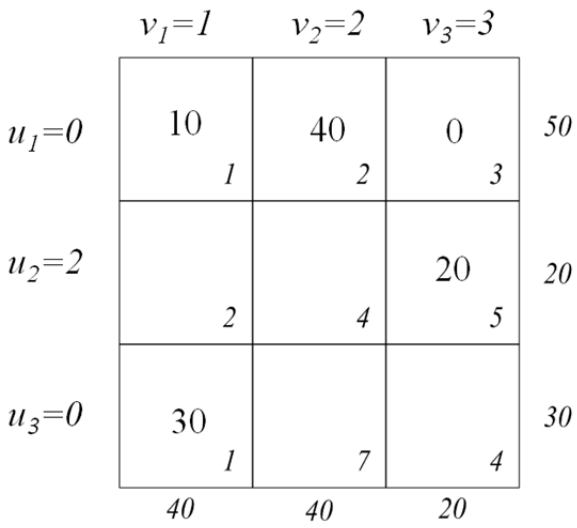
(iii) se $x_{ij} = 0$ então $u_i + v_j \leq c_{ij}$

A primeira condição garante a admissibilidade primal, as duas outras condições garantem o respeito do teorema da folga complementar e a admissibilidade dual.

Algoritmo

O algoritmo simplex para transportes é um algoritmo primal. Parte de uma solução (básica) admissível primal, com base nela e na condição (ii) obtém uma solução dual, caso a condição (iii) não se verifique passa para outra base admissível com menor custo. Exemplifica-se agora o algoritmo. A solução inicial tem um custo de 220.

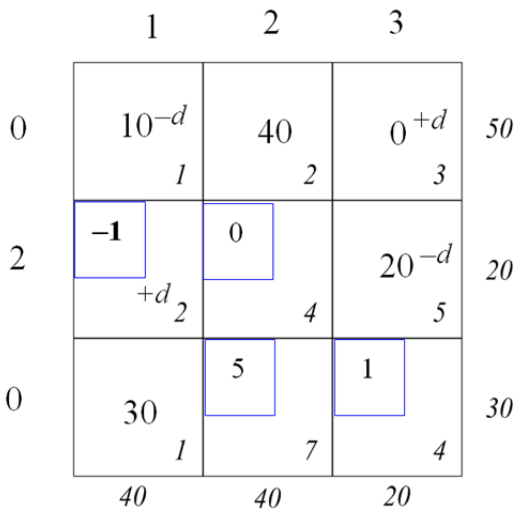
Com base em, para cada variável básica x_{ij}, se verificar $u_i + v_j = c_{ij}$, é possível obter uma solução dual que é representada no seguinte quadro:



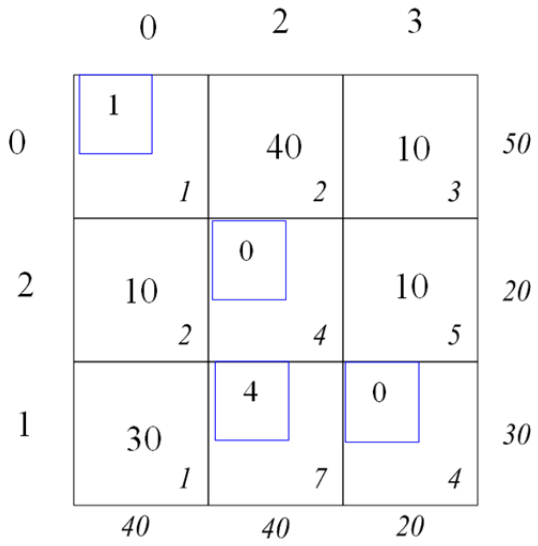
Esta solução foi obtida arbitrando $u_1 = 0$. Como x_{11} é básica, $u_1 + v_1 = c_{11}$. Sendo $u_1 = 0$ e $c_{11} = 1$, $v_1 = 1$. Com base nas restantes células básicas (31,12,13,23) é possível determinar os valores de todas as variáveis duais.

A solução é óptima se o valor $c_{ij} - u_i - v_j$ fôr maior ou igual a zero para todas as variáveis não básicas. Esse valor é o custo reduzido. Se todos os custos reduzidos forem não negativos, a solução é óptima. Se não, escolhe-se a variável com custo reduzido mais negativo para entrar na base com valor $+d$.

No exemplo, os custos reduzidos estão representados no canto superior esquerdo. O menor (que é negativo) é o da célula 21 que entra na base. Ao ser aumentado o fluxo nessa célula, tem de diminuir na 11 e na 23. Para o número de unidades nas origens e destinos permanecer constante tem de aumentar na célula 13. O aumento do fluxo na célula 21 é limitado pela célula 11, logo $d = 10$.



A solução para a próxima iteração é:



Esta solução é óptima porque todos os custos reduzidos são não negativos. O seu custo é $220-10=210$.

Resumo

Resume-se agora o algoritmo simplex para transportes.

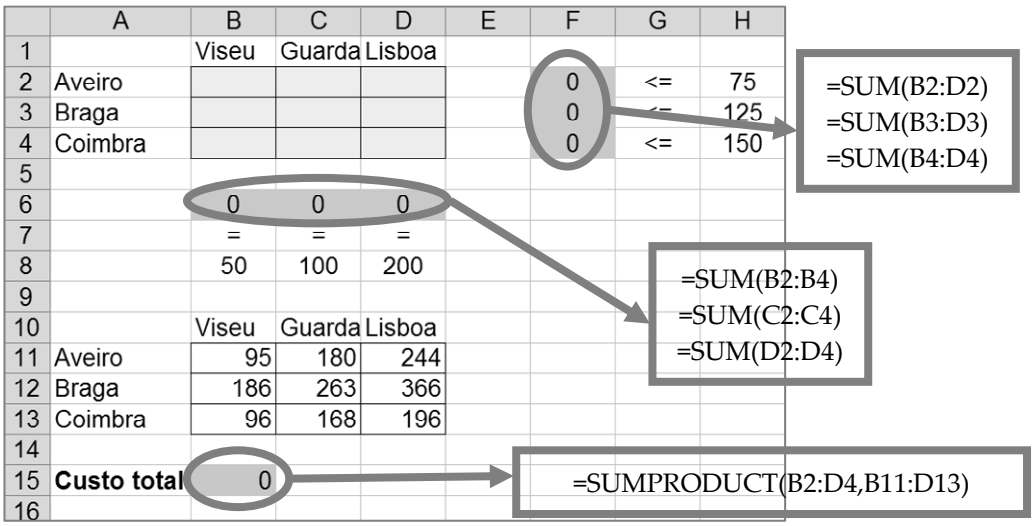
1. Se a oferta total não é igual à procura total, introduzir uma origem (destino) fictícia(o) com a oferta (procura) adequada. No caso de existirem pares origem-destino entre os quais não é possível haver transportes, podem na mesma ser considerados com um custo muito elevado (usualmente representado por M à semelhança do método do grande M). Depois de se ter uma solução admissível (nenhuma variável básica tem custo M) pode deixar de calcular-se o custo reduzido desses arcos.
2. Obter uma solução básica admissível inicial através do método dos custos mínimos. Verificar que o número de variáveis básicas é igual ao número de origens mais destinos menos um.
3. Arbitrando $u_1 = 0$, determinar os valores das variáveis duais tal que, para cada célula básica, $u_i + v_j = c_{ij}$.
4. Calcular o custo reduzido de cada célula não básica $c_{ij} - u_i - v_j$.
5. Se os custos reduzidos são todos não negativos, parar: a solução actual é óptima. Caso contrário, seleccionar a célula com o custo reduzido mais negativo (em caso de empate, escolher uma arbitrariamente) para entrar na base. Designa-se essa célula por kl. Identificar o ciclo de células básicas formado pela inclusão na base do arco kl. Existe sempre um e um só ciclo.
6. Considerar um fluxo de d unidades no arco kl. Determinar o valor máximo que d pode tomar tendo em conta que um fluxo de d no arco kl implica um aumento de d unidades no fluxo dos arcos com o mesmo sentido de kl e uma diminuição de d unidades no fluxo dos arcos com o sentido contrário ao de kl. O arco que sai da base é aquele cujo fluxo primeiro se anula com o aumento de fluxo em kl.

7. Actualizar a solução primal. Ir para 3.

5.3.4 Transportes numa folha cálculo

Dada a sua estrutura, é extremamente simples obter a solução óptima de um problema de transportes numa folha de cálculo (com um *solver* de programação linear).

Na Figura 5.14 apresenta-se um modelo de transportes. As variáveis de decisão correspondem às células B2:D4. As restrições são F2:F4=H2:H4 e B6:D6=B8:D8. A função objectivo corresponde à célula B15.



	A	B	C	D	E	F	G	H
1		Viseu	Guarda	Lisboa				
2	Aveiro					0	<=	75
3	Braga					0	<=	125
4	Coimbra					0	<=	150
5								
6		0	0	0				
7		=	=	=				
8		50	100	200				
9								
10		Viseu	Guarda	Lisboa				
11	Aveiro	95	180	244				
12	Braga	186	263	366				
13	Coimbra	96	168	196				
14								
15	**Custo total**	0						
16								

Figura 5.14. Transportes numa folha de cálculo.

Após inserir as referências nas janelas do *solver* e optimizado, deve-se ter o cuidado de verificar que a mensagem retornada pelo *solver* indica que foi obtida uma solução óptima. Na Figura 5.15 é apresentada a folha de cálculo com a solução óptima.

	A	B	C	D	E	F	G	H	I
1		Viseu	Guarda	Lisboa					
2	Aveiro	25	0	50		75	<=	75	
3	Braga	25	100	0		125	<=	125	
4	Coimbra	0	0	150		150	<=	150	
5									
6		50	100	200					
7		=	=	=					
8		50	100	200					
9									
10		Viseu	Guarda	Lisboa					
11	Aveiro	95	180	244					
12	Braga	186	263	366					
13	Coimbra	96	168	196					
14									
15	**Custo total**	74925							
16									

Figura 5.15. Folha de cálculo após optimização.

5.4 Fluxo máximo

Num problema de fluxo máximo, pretende-se determinar a maior quantidade de fluxo que é possível enviar entre dois nodos (fonte e poço) de uma rede em que os arcos têm capacidades.

Exemplo 5.3 Fluxo máximo

Considere o problema de determinar a taxa de evacuação de uma sala de espectáculos com a configuração representada na Figura 5.16, onde a área a evacuar está assinalada a cinzento. As diferentes portas estão representadas por letras associadas a diferentes capacidades (A – 600 pessoas por minuto; B – 400 pessoas por minuto; C – 800 pessoas por minuto). O corredor de passagem entre as portas A e B tem capacidade para 350 pessoas por minuto em cada sentido. O modelo de fluxo máximo para este problema é dado na Figura 5.17.

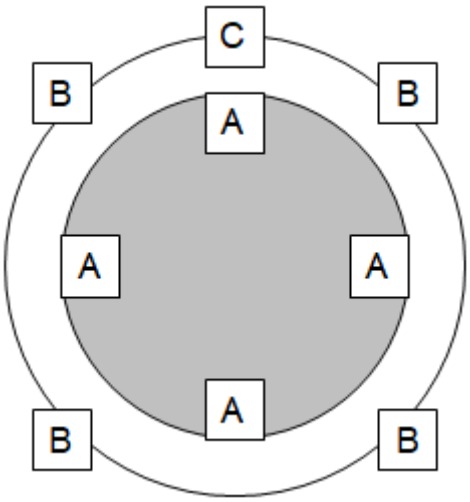


Figura 5.16. Planta da sala de espectáculos do problema de fluxo máximo.

Os parâmetros de um modelo de fluxo máximo são um grafo, os nodos onde o fluxo tem origem (s) e onde o fluxo tem destino (t), as capacidades dos arcos ($u_{ij}, \forall ij \in A$).

Um modelo de programação linear baseia-se nas variáveis de decisão

v – fluxo entre s e t

x_{ij} – fluxo no arco ij, $\forall ij \in A$

O modelo geral é

$$Max\ v$$

$$sujeito\ a:$$

$$\sum_{j:ij \in A} x_{ij} - \sum_{j:ji \in A} x_{ij} = \begin{cases} v\ para\ i = s \\ 0\ para\ i \neq s\ e\ i \neq t \\ -v\ para\ i = t \end{cases}$$

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

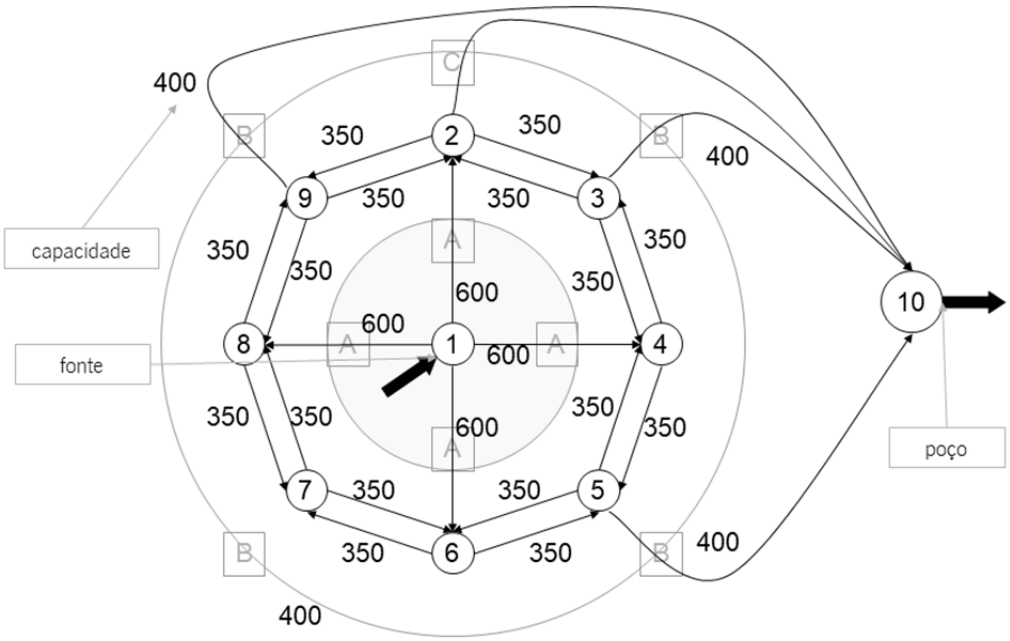


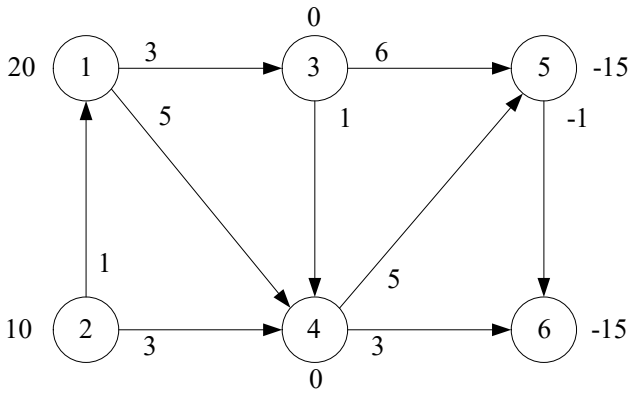
Figura 5.17. Modelo do exemplo de problema de fluxo máximo.

■

5.5 Exercícios

Exercício 5.1 Distribuição

Uma determinada empresa de distribuição tem um produto armazenado em duas localizações distintas. Na localização 1 tem 20 unidades e na localização 2 tem 10 unidades. Dois clientes, localizados em 5 e 6, pretendem adquirir, cada um, 15 unidades do produto em causa. Na Figura representam-se todas as localizações relevantes deste problema, bem como o custo associado a transportar uma unidade do produto entre aquelas em que é possível fazê-lo. Pretende-se determinar a forma mais económica de transportar as 30 unidades com o menor custo possível, sabendo que não é possível transportar mais de 15 unidades entre cada par de localizações.



a) Apresente um modelo de programação linear para este problema.

b) Obtenha uma solução óptima utilizando um *solver* de programação linear.

Exercício 5.2 Distribuição

Uma empresa produz um mesmo produto em duas diferentes fábricas. Depois de fabricado o produto é enviado para dois armazéns. Entre a primeira fábrica e o primeiro armazém é possível, por comboio, transportar uma qualquer quantidade do produto. O mesmo se passa entre a segunda fábrica e o segundo armazém. É também possível efectuar o transporte, através de camiões, das fábricas para um centro de distribuição e deste para os armazéns. Entre cada fábrica e o centro de distribuição é possível, no máximo, transportar 50 unidades. Entre o centro de distribuição e cada armazém é possível, no máximo, transportar 60 unidades.

Na tabela representam-se os custos unitários de transporte para as diversas alternativas juntamente com as quantidades a serem produzidas em cada fábrica e a serem armazenadas em cada armazém.

	Centro de Distribuição	Armazém 1	Armazém 2	Quantidade a produzir
Fábrica 1	3	7	–	80
Fábrica 2	4	–	9	70
Centro de Distribuição	–	2	4	–
	Quantidade a armazenar	60	90	

a) Apresente uma representação em rede do modelo de fluxo de custo mínimo para este problema.

b) Apresente o modelo de programação linear correspondente.

c) Obtenha uma solução por inspecção, justificando o seu raciocínio.

d) Obtenha uma solução óptima utilizando um *solver* de programação linear.

Exercício 5.3 Produção e distribuição

Uma empresa tem duas fábricas (F1, F2) e dois centros de distribuição (CD1, CD2) que servem três regiões (R1, R2, R3). Numa dada semana, existem encomendas de 10, 15 e 20 mil unidades nas regiões R1, R2 e R3. Cada fábrica tem capacidade de produzir 30 mil unidades. Considere que é possível transportar unidades de qualquer fábrica para qualquer centro de distribuição e de qualquer centro de distribuição para qualquer região com os custos unitários dados nas duas tabelas abaixo.

	CD1	CD2
F1	125	151
F2	134	142

	R1	R2	R3
CD1	62	75	82
CD2	96	81	72

a) Apresente uma representação em rede do modelo de fluxo de custo mínimo para este problema.

b) Apresente o modelo de programação linear correspondente.

c) Obtenha uma solução óptima utilizando um *solver* de programação linear.

Exercício 5.4 Planeamento de produção – Fluxo de custo mínimo

Considere um problema de planeamento de produção de um artigo para seis semanas. Para satisfazer as encomendas de uma determinada semana, a fábrica pode produzir durante a própria semana ou numa semana anterior. Assim, consideram-se custos unitários de produção e custos unitários de armazenamento (por cada semana que a unidade está armazenada). Quando há a produção para uma semana é feita numa semana anterior, as unidades ficam armazenadas até ao final da semana em que têm de ser entregues. Na tabela, são dadas as estimativas, para cada semana, da procura, dos custos unitários de produção e dos custos unitários de armazenamento. O objectivo é minimizar o custo total.

Semana	1	2	3	4	5	6
Procura	101	122	147	103	135	122
Custo unitário de produção	5	1	3	2	4	5
Custo unitário de armazenamento	1	1	2	2	3	4

a) Apresente uma representação em rede do modelo de fluxo de custo mínimo para este problema.

b) Apresente o modelo de programação linear correspondente.

c) Como incorporaria nos modelos em cada mês não ser possível produzir mais de 130 unidades e em cada mês não ser possível armazenar mais de 40 unidades.

d) Partindo do problema da alínea anterior, considere agora que é possível entregar unidades com atraso. Assim, a fábrica pode produzir numa semana posterior para satisfazer a procura de uma determinada semana. O custo unitário do atraso por semana é o dobro do custo unitário de armazenamento na mesma semana.

e) Obtenha uma solução óptima para cada uma das três alíneas anteriores, utilizando um *solver* de programação linear.

Exercício 5.5 Planeamento de produção – Transportes

Considere um problema de planeamento de produção de um artigo para seis semanas. Para satisfazer as encomendas de uma determinada semana, a fábrica pode produzir durante a própria semana ou numa semana anterior. Assim, consideram-se custos unitários de produção e custos unitários de armazenamento (por cada semana que a unidade está armazenada). Quando há a produção para uma semana é feita numa semana anterior, as unidades ficam armazenadas até ao final da semana em que têm de ser entregues. Na tabela, são dadas as estimativas, para cada semana, da procura, dos custos unitários de produção e dos custos unitários de armazenamento. O objectivo é minimizar o custo total.

Semana	1	2	3	4	5	6
Procura	101	122	147	103	135	122
Custo unitário de produção	5	1	3	2	4	5
Custo unitário de armazenamento	1	1	2	2	3	4

a) Apresente uma representação em rede do modelo de transportes para este problema.

b) Apresente o modelo de programação linear correspondente.

c) Como incorporaria nos modelos em cada mês não ser possível produzir mais de 130 unidades e em cada mês não ser possível armazenar mais de 40 unidades.

d) Partindo do problema da alínea anterior, considere agora que é possível entregar unidades com atraso. Assim, a fábrica pode produzir numa semana posterior para satisfazer a procura de uma determinada semana. O custo unitário do atraso por semana é o dobro do custo unitário de armazenamento na mesma semana.

e) Obtenha uma solução óptima para cada uma das três alíneas anteriores, utilizando um *solver* de programação linear.

Exercício 5.6 Viagem aérea

Uma determinada companhia aérea pondera a criação de um voo diário que tenha início em Bragança, pare no Porto, depois em Lisboa e termine em Faro. Em cada cidade podem entrar ou sair passageiros do avião. Este tem uma capacidade de *100* lugares. O número de pessoas que pretende viajar entre cada par de cidades e o preço de cada viagem são os dados na tabela.

A companhia aérea pretende determinar o número de lugares que deve reservar para cada viagem entre cidades de forma a maximizar o lucro.

	BF	BL	BP	PF	PL	LF
Preço	200	150	100	175	75	125
Procura	20	45	10	40	90	55

a) Apresente uma representação em rede do modelo de fluxo de custo mínimo para este problema.

b) Apresente o modelo de programação linear correspondente.

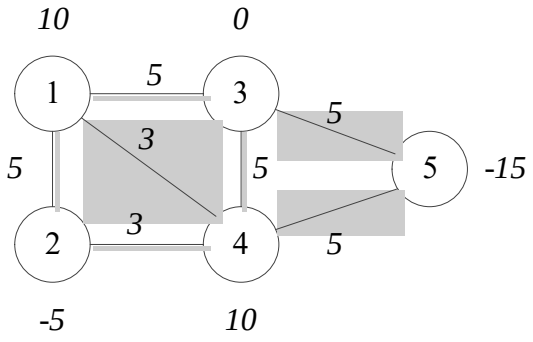
c) Obtenha uma solução óptima utilizando um *solver* de programação linear.

c) Interprete a solução obtida. Qual o valor do lucro total?

[adaptado de R. Ahuja, T. Magnanti, J. Orlin, "Network Flows", 1993, Prentice-Hall]

Exercício 5.7 Fluxo em rede não orientada com capacidades

Considere a seguinte rede em que os valores juntos aos arcos correspondem aos custos associados a uma unidade de fluxo em qualquer dos sentidos e os valores junto aos nodos correspondem às suas ofertas/procuras. A capacidade (considerando os fluxos em ambos os sentidos) das arestas {3,5} e {4,5} é 10. A capacidade das restantes é 5. No nodo 3 apenas podem passar 5 unidades de fluxo.



a) Apresente um modelo de programação linear que permita a minimização do custo total.

b) Obtenha uma solução óptima utilizando um *solver* de programação linear.

Exercício 5.8 Representação de uma rede com base em restrições lineares

Represente a rede associada às restrições de um modelo de programação linear apresentadas de seguida.

$x_{12} + x_{13} + x_{14} = 2$

$-x_{12} + x_{24} = 1$

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

$-x_{14} - x_{24} + x_{45} + x_{46} = 0$

$-x_{35} - x_{45} + x_{56} = 0$

$-x_{46} - x_{56} = -3$

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

Exercício 5.9 Fluxo de custo mínimo em folha de cálculo

Na figura é apresentado um modelo de fluxo de custo mínimo resolvido no solver de uma folha de cálculo.

a) Represente o mesmo modelo numa rede.

b) Indique o significado dos valores das células C2:J2 e da célula L11.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2			20	0	0	30	0	30	0	25				
3														
4			x12	x13	x14	x23	x25	x34	x35	x45				
5		1	1	1	1							20	=	20
6		2	-1			1	1					10	=	10
7		3		-1		-1		1	1			0	=	0
8		4			-1			-1		1		-5	=	-5
9		5					-1		-1	-1		-25	=	-25
10														
11			4	5	6	7	2	4	6	7		585		

Exercício 5.10 Serviço docente

O Departamento de uma Universidade está a preparar o próximo ano lectivo, sendo necessário definir entre três docentes (A, B e C) o número de horas por semana que cada um deve dar a seis unidades curriculares (UCs). Cada docente tem de dar 8 horas e cada UC tem 4 horas por semana. De acordo com as suas formações, para cada UC, os docentes foram ordenados do mais adequado (valor 1) ao menos adequado (valor 3). A tabela seguinte reflecte essa ordenação.

	1	2	3	4	5	6
A	1	1	2	1	1	3
B	2	3	1	2	3	2
C	3	2	3	3	2	1

a) Apresente um modelo de programação linear para este problema.

b) Obtenha uma solução óptima utilizando um *solver* de programação linear.

Exercício 5.11 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 Centros de Saúde 1, 2 e 3 e pelos preços de 0.2€, 0.4€ e 0.3€.

a) Apresente um modelo de programação linear para este problema.

b) Qual a forma mais económica de resolver o problema em causa? Utilize um *solver* de programação linear.

Exercício 5.12 Realocação de alunos

Foi decidido que, no próximo ano lectivo, uma determinada escola deveria ser encerrada. Dessa forma, os alunos actualmente a estudar nessa escola, terão de passar a estudar em uma de três outras escolas.

Dado o transtorno provocado, foi decidido que, para os alunos em questão, seria providenciado o transporte entre a sua área de residência e a sua nova escola. Consideraram-se seis áreas de residência.

Na seguinte Tabela apresentam-se alguns dados referentes a esta situação.

Área	Número de alunos	Custo de transporte por estudante (€)		
		Escola 1	Escola 2	Escola 3
1	450	300	0	700
2	600	–	400	500
3	550	600	300	200
4	350	200	500	–
5	500	0	–	400
6	450	500	300	0
Capacidade da escola		900	1100	1000

a) Apresente um modelo de programação linear para este problema.

b) Por inspecção, indique qual a forma mais económica de resolver o problema.

Exercício 5.13 Tratamento de resíduos sólidos

Uma empresa de valorização e tratamento de resíduos sólidos pretende proceder à recolha de resíduos de três locais (A, B e C) e transportá-los para três estações de tratamento (E1, E2 e E3). As quantidades existentes nos locais são 3, 12 e 20 toneladas, pretendendo-se que a estação E1 receba 13 toneladas, a estação E2 receba 8 toneladas e a estação E3 receba 14 toneladas. Os custos (em Unidades Monetárias por tonelada) de transporte entre cada local e cada estação são dados na tabela seguinte.

	E1	E2	E3
A	1	2	3
B	4	5	6
C	7	10	10

Qual a melhor forma de transportar os resíduos para as estações de tratamento e qual o custo que lhe está associado? Utilize um algoritmo adequado.

Exercício 5.14 Algoritmo de transportes

Considere um seguinte problema de transportes com três origens (O1, O2 e O3) e três destinos (D1, D2 e D3). A oferta de cada uma das origens é de 15 unidades e as procuras de cada um dos destinos é de 10 unidades.

Nas duas tabelas seguintes apresentam-se os custos unitários de transporte entre cada origem e cada destino e uma solução admissível para o problema em causa.

Custos unitários	D1	D2	D3
O1	7	4	3
O2	3	1	7
O3	1	6	1

Solução	D1	D2	D3
O1	0	0	5
O2	0	10	0
O3	10	0	5

a) Através do cálculo dos custos reduzidos, mostre que a solução apresentada é óptima.

b) Existem soluções óptimas alternativas? Justifique e, em caso afirmativo, indique uma.

c) Obtenha uma solução óptima, considerando agora que o transporte entre O1 e D3 não é permitido.

Exercício 5.15 Tarefas e máquinas

Considere o problema de seleccionar quais as máquinas que devem executar um conjunto de tarefas. O parque de máquinas é constituído por um total de 15 máquinas, de 3 tipos diferentes. Há 5 máquinas de cada um dos 3 tipos. As tarefas a realizar são de 4 tipos diferentes. É necessário realizar a tarefa 1, uma única vez. A tarefa 2 tem de ser realizada 2 vezes. As tarefas 3 e 4 têm de ser realizadas 6 vezes. Os lucros associados à realização de cada tarefa por cada tipo de máquina são os dados no quadro seguinte.

		Tipo de tarefa			
		1	2	3	4
Tipo de máquina	1	5	3	4	2
	2	9	8	8	4
	3	10	4	8	7

Utilizando um algoritmo adequado, determine em que máquinas devem ser feitas as tarefas.

Exercício 5.16 Produção em fábricas

Uma empresa adquiriu três fábricas (F1, F2 e F3) aptas a iniciar a produção de três novos produtos (P1, P2 e P3). A tabela seguinte indica a capacidade de produção mensal disponível em cada fábrica, a procura mensal de cada produto e os custos unitários (em €) de fabrico de cada produto em cada fábrica.

	P1	P2	P3	Capacidade disponível
F1	6	4	2	10
F2	6	3	4	20
F3	7	10	4	15
Procura a satisfazer	5	18	22	

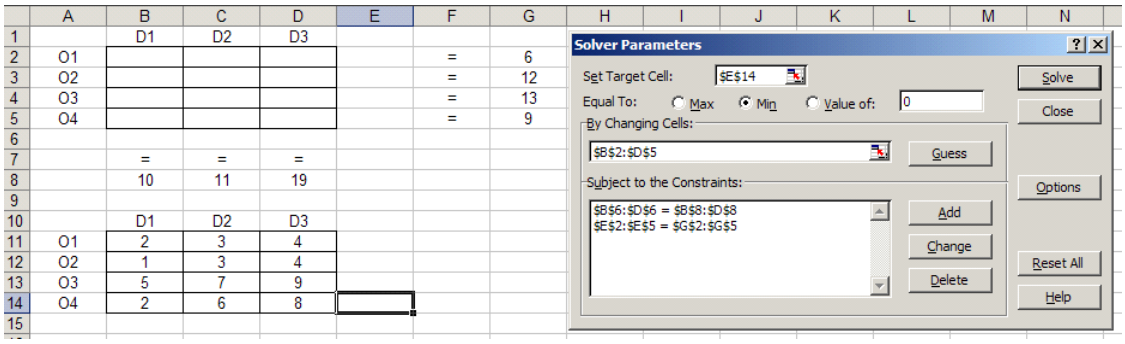
Uma proposta de distribuição da produção pelas fábricas é a seguinte: 3 unidades do produto 1 são fabricadas em F1 e 2 unidades do mesmo produto são fabricadas em F2; as 18 unidades do produto 2 são fabricadas em F2; 7 unidades do produto 3 são fabricadas em F1 e 15 unidades do mesmo produto são fabricadas em F3.

a) Indique o custo dessa proposta.

b) Através de um algoritmo adequado, obtenha uma proposta com menor custo.

Exercício 5.17 *Solver* do *Excel* + algoritmo de transportes

Na resolução de um modelo de transportes foi utilizado o *Solver* do *Excel*, tal como mostrado na figura. Indique quais as fórmulas que deverão constar nas células referenciadas na caixa de diálogo "*Solver Parameters*".



Obtenha a solução óptima e o seu valor utilizando o algoritmo de transportes. Utilize a seguinte solução inicial: as 6 unidades de O1 são transportadas para D2; as 12 unidades de O2 são transportadas para D3; 1 unidade de O3 é transportada para D1; 5 unidades de O3 são transportadas para D2; 7 unidades de O3 são transportadas para D3 e as 9 unidades de O4 são transportadas para D1.

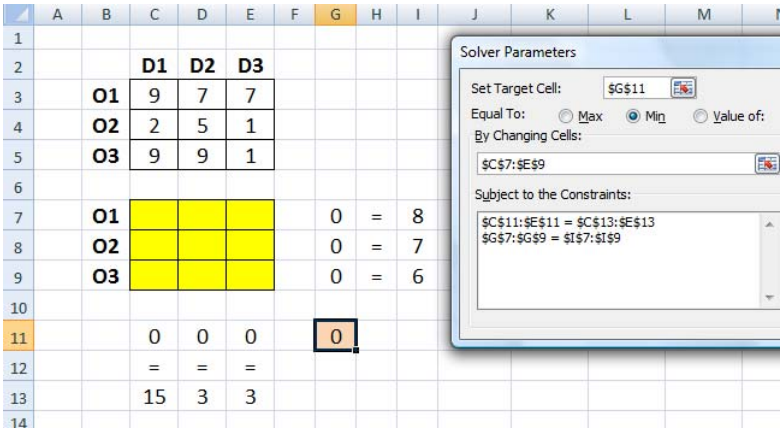
Exercício 5.18 *Solver* do *Excel* + programação linear + algoritmo de transportes

Na resolução de um problema de transportes com o *Solver* de uma folha de cálculo, introduziu-se o modelo apresentado na figura seguinte.

a) Qual a fórmula da célula G11?

b) Apresente o modelo de programação linear para este problema.

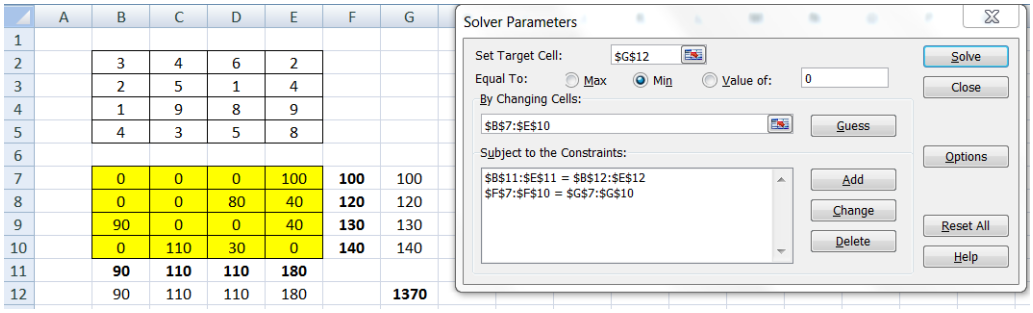
c) Partindo da solução em que 5 unidades de O1, 4 unidades de O2 e 6 unidades de O3 são enviadas para o destino D1, 3 unidades de O1 são enviadas para D2 e 3 unidades de O2 são enviadas para D3, obtenha uma solução óptima.



		D1	D2	D3			
O1	9	7	7	0	=	8
O2	2	5	1	0	=	7
O3	9	9	1	0	=	6
	0	0	0	0		
	=	=	=			
	15	3	3			

Exercício 5.19 Solver do excel

A imagem seguinte mostra um modelo de transportes como introduzido numa folha de cálculo e a correspondente caixa de diálogo do *solver*. A solução apresentada é uma solução óptima.



3	4	6	2		
2	5	1	4		
1	9	8	9		
4	3	5	8		
0	0	0	100	**100**	100
0	0	80	40	**120**	120
90	0	0	40	**130**	130
0	110	30	0	**140**	140
90	**110**	**110**	**180**		
90	110	110	180		**1370**

Indique:

a) As células que correspondem a variáveis de decisão.

b) As fórmulas que foram inseridas nas células F7 a F10.

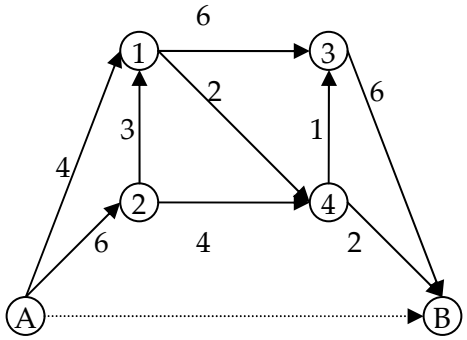
c) As fórmulas que foram inseridas nas células B11 a E11.

d) A fórmula que foi inserida na célula G12.

e) Qual o valor óptimo?

Exercício 5.20 Percurso alternativo

Considere a representação de um conjunto de estradas de sentido único da Figura e que, por motivo de obras, se pretende cortar o tráfego na estrada AB, desviando-o para as restantes estradas. Junto a cada arco indica-se o número de veículos que podem atravessar a estrada a ele associada (em centenas de veículos por hora).



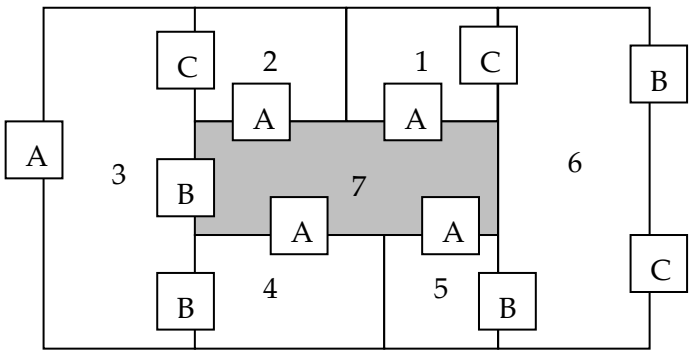
Qual o número máximo de veículos que poderão ser desviados por hora?

Exercício 5.21 Evacuação de edifício

a) Considere o problema de determinar a taxa de evacuação de uma sala de espectáculos com a configuração representada na Figura, onde a área a evacuar está assinalada a cinzento e com o número 7.

As diferentes portas estão representadas por letras associadas a diferentes capacidades (A – 80 pessoas por minuto; B – 90 pessoas por minuto; C – 100 pessoas por minuto).

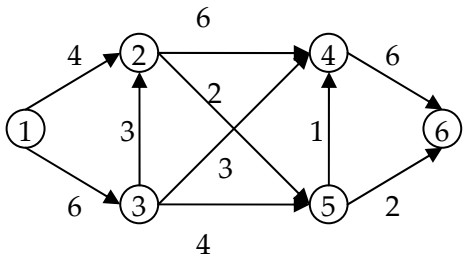
Apresente um modelo de fluxo máximo para este problema.



b) Apresente o modelo geral de Programação Linear para o problema de fluxo de custo mínimo utilizando a seguinte notação: A e N conjunto de arcos e conjunto de nodos, respectivamente; u_{ij} capacidade do arco ij, $\forall ij \in A$; c_{ij} custo unitário do arco ij, $\forall ij \in A$; b_i oferta/procura do nodo i, $\forall i \in N$.

Exercício 5.22 Modelo de rede geral

Considere a rede representada de seguida.



a) Determine o fluxo máximo entre 1 e 6, considerando que os valores juntos aos arcos correspondem a capacidades.

b) Determine o fluxo de custo mínimo, considerando que os valores juntos aos arcos correspondem a custos unitários, que as ofertas dos nodos 1 e 2 são de 10 e 15 unidades, respectivamente e que as procuras dos nodos 4 e 6 são de 12 e 13 unidades, respectivamente.

c) Determine o fluxo de custo mínimo nas circunstâncias da alínea anterior, considerando também que o limite superior de cada arco é de 10 unidades.

Exercício 5.23 Modelo de rede geral

Uma empresa tem duas fábricas (F1 e F2) onde pode produzir dois modelos (P1 e P2) de um automóvel. As capacidades de produção (em unidades por semana) de cada modelo em cada fábrica são dadas na tabela, assim como o custo de produção (em U.M./unidade). A fábrica F1 pode produzir no máximo 800 unidades. A fábrica F2 pode produzir no máximo 700 unidades.

A empresa vende os automóveis em duas regiões (C1 e C2), sendo os custos de transporte (em U.M./unidade) de cada produto para cada região dados na mesma tabela. A empresa pretende determinar a forma mais económica de satisfazer a procura de 200 unidades do modelo 1 e 300 unidades do modelo 2 por semana na região 1, e 400 e 200 dos modelos 1 e 2 na região 2. Os custo de transopre são dados na segunda tabela.

	Capacidade de produção		Custo de produção	
	P1	P2	P1	P2
F1	400	300	13	12
F2	350	200	16	11

Custo de transporte	P1		P2	
	C1	C2	C1	C2
F1	2	3	5	1
F2	4	3	2	3

Exercício 5.24 Fluxo min max

Considere uma rede com ofertas e procuras associados aos nodos e capacidades associadas aos arcos. Sabendo que a carga de um arco é a proporção da capacidade do arco que é usada (carga=fluxo/capacidade), apresente um modelo de programação linear que minimize a carga do arco com maior carga.

5.6 Bibliografia

- R. Ahuja, T. Magnanti, J. Orlin, "Network Flows", 1993, Prentice-Hall.
- Schrijver, A. (2002). Combinatorial optimization: polyhedra and efficiency (Vol. 24). Springer Science & Business Media.

5.7 Resultados de aprendizagem

- Representar problemas de fluxo de custo mínimo, transportes e fluxo máximo através de modelos de programação inteira.
- Resolver os mesmos problemas com o Solver para Excel.
- Aplicar algoritmos específicos para resolver instâncias de pequena dimensão de problemas de fluxo de custo mínimo e transportes.
- Modelar problemas de optimização através de fluxos em rede.

6
Fluxo multicomodidade

6.1 Introdução

Os modelos de fluxo multicomodidade permitem abordar problemas em que os recursos de uma rede, em particular as capacidades dos arcos, são partilhados por entidades (ou comodidades / "commodities") diferentes.

Estes problemas têm sido estudados desde os finais dos anos 1950, sendo um dos primeiros trabalhos relevantes o de Ford e Fulkerson (1958). Existem diversos artigos de revisão dos primeiros trabalhos como o de Assad (1978) e Kennington (1978). Mais recentemente o livro essencial na áreas de fluxos em rede de Ahuja, Magnanti e Orlin (1993) inclui um capítulo dedicado a fluxos multicomodidade.

Problemas e modelos de fluxo multicomodidade surgem em diversas aplicações, sendo particularmente relevantes em produção, logística e redes de computadores.

6.2 Definição do problema base

Formalmente, os parâmetros que definem um problema de fluxo multicomodidade (em que cada comodidade tem uma origem e um destino) são os seguintes:

- $G = (N, A)$ grafo orientado
- h número de comodidades
- K conjunto de comodidades
- o^k nodo origem da comodidade k, $k \in K$
- d^k nodo destino da comodidade $k, k \in K$
- b^k procura da comodidade k, $k \in K$
- c_{ij}^k custo unitário do arco de i para j para a comodidade k, $\forall ij \in A, \forall k \in K$

A procura de uma comodidade é uma medida do fluxo que tem de existir entre a sua origem e o seu destino. Em redes de computadores tipicamente corresponde ao volume de tráfego médio ou de pico estimado no sentido da origem para o destino (expresso em Mbps, mega bits por segundo, ou pps, pacotes por segundo). Numa rede logística, a procura corresponde normalmente às quantidades requeridas nos nodos associados a clientes.

Se o problema fosse o de encaminhar as comodidades com o menor custo possível sem restrições adicionais poderia ser resolvido através de problemas de caminho mais curto independentes (um para cada comodidade). Nesse caso não haveria partilha de

recursos. Na versão base abordada de seguida, o que faz com que o problema não possa ser resolvido dessa forma é existirem capacidades associadas aos arcos – a quantidade total que passa em cada arco é limitada por um determinado valor.

Formalmente:

- u_{ij} capacidade do arco $ij, \forall ij \in A$.

Excepto quando explicitamente indicado, assume-se que o fluxo de cada comodidade é divisível (pode usar mais do que um caminho).

Descreve-se agora uma instância de um problema de fluxo multicomodidade definida sobre a rede da Figura 6.1. Consideram-se duas comodidades, a primeira com origem em 1, destino 5 e procura 5, e a segunda com origem em 2, destino 5 e procura 4. Os custos unitários são representados junto a cada arco. A capacidade de cada arco é de 6.

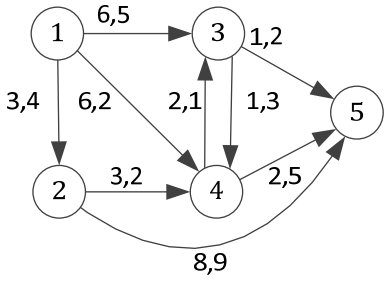


Figura 6.1. Parte de uma instância do problema de fluxo multicomodidade.

Uma solução óptima para esta instância é 2 unidades da comodidade 1 percorrerem os arcos 1-3 e 3-5, 3 unidades percorrerem os arcos 1-4 e 4-5 e as 4 unidades da comodidade 2 percorrerem os arcos 2-4, 4-3 e 3-5.

Na maior parte deste capítulo aborda-se o caso agora descrito em que cada comodidade tem uma origem e um destino (a excepção é a secção 6.8 em que uma comodidade tem múltiplas origens e múltiplos destinos). O caso em que uma comodidade tem uma origem e múltiplos destinos ou um destino e múltiplas origens pode facilmente ser convertido no caso de uma origem e um destino (e vice-versa).

6.3 Modelos

6.3.1 Modelo de arcos

Um modelo para o problema de fluxo multicomodidade base baseia-se em variáveis de decisão associadas a fluxos nos arcos:

- x_{ij}^k – proporção em relação à procura de fluxo no arco ij da comodidade k, $\forall ij \in A, \forall k \in K$

Modelo de programação linear:

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

$$sujeito\ a:$$

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

$$\sum_{k \in K} b^k x_{ij}^k \leq u_{ij}, \forall ij \in A$$

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

O primeiro grupo de restrições conserva o fluxo de cada comodidade em cada nodo e o segundo grupo de restrições impede que a capacidade dos arcos seja excedida.

6.3.2 Modelo de caminhos

Um modelo alternativo para o problema de fluxo multicomodidade base baseia-se em variáveis de decisão associadas a fluxos nos caminhos. Neste modelo, o primeiro passo é a enumeração de todos os caminhos para todas as comodidades (se se pretender a solução óptima) ou um subconjunto de caminhos (se uma solução aproximada for suficiente). Os caminhos considerados designam-se por caminhos candidatos.

É de salientar a definição dos conjuntos candidatos pode por si só incorporar restrições. É possível, por exemplo, considerar apenas caminhos com um número de saltos menor do que um determinado valor.

Representa-se por P^k o conjunto dos caminhos candidatos da comodidade k. A cada caminho candidato p de uma comodidade k associa-se uma variável de decisão:

- x^{pk} – proporção (em relação à procura) no caminho candidato p da comodidade $k, \forall k \in K, \forall p \in P^k$

Considera-se o parâmetro adicional:

- $a_{ij}^{pk} = \begin{cases} 1, se\ caminho\ p\ da\ comodidade\ k\ inclui\ o\ arco\ ij \\ 0, caso\ contrário \end{cases} \forall k \in K, \forall p \in P^k, \forall ij \in A$

e define-se o custo de um caminho p da comodidade k por:

$$c^{pk} = \sum_{ij \in A} c_{ij}^k a_{ij}^{pk}$$

Modelo de programação inteira:

$$Min\ z = \sum_{k \in K} \sum_{p \in P^k} c^{pk} x^{pk}$$

$$sujeito\ a:$$

$$\sum_{p \in P^k} x^{pk} = 1$$

$$\sum_{k \in K} \sum_{p \in P^k} a_{ij}^{pk} b^k x^{pk} \leq u_{ij}, \forall ij \in A$$

$$x^{pk} \geq 0, \forall k \in K, \forall p \in P^k$$

A título exemplificativo, considere-se uma instância com duas comodidades: uma com origem 1, destino 4 e procura 3 e outra com origem 1, destino 5 e procura 2. A capacidade de todos os arcos é 4, $u_{ij} = u = 4, \forall ij \in A$. Os custos são independentes das comodidades, $c_{ij}^k = c_{ij}, \forall ij \in A$, e são dados na figura.

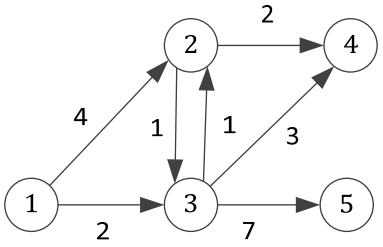


Figura 6.2. Parte de uma instância do problema de fluxo multicomodidade.

Os caminhos entre 1 e 4 ($k = 1$) são $1-2-3-4$ ($p = 1$), $1-2-4$ ($p = 2$), $1-3-2-4$ ($p = 3$) e $1-3-4$ ($p = 4$). Os caminhos entre 1 e 5 ($k = 2$) são $1-2-3-5$ ($p = 1$) e $1-3-5$ ($p = 2$).

O modelo de programação inteira é assim:

$$Min\ z = 8x^{11} + 6x^{21} + 5x^{31} + 8x^{41} + 12x^{12} + 9x^{22}$$

$sujeito\ a$:

$$x^{11} + x^{21} + x^{31} + x^{41} = 1$$

$$x^{12} + x^{22} = 1$$

$$3x^{11} + 3x^{21} + 2x^{12} \leq 4$$

$$3x^{31} + 3x^{41} + 2x^{22} \leq 4$$

...

$$2x^{12} + 2x^{22} \leq 4$$

$$x^{pk} \geq 0, \forall k \in K, \forall p \in P^k$$

É de notar a relação entre as variáveis de decisão de ambos os modelos (arcos e caminhos) que se traduz em

$$x_{ij}^k = \sum_{p \in P^k} a_{ij}^{pk} x^{pk}, \forall ij \in A, \forall k \in K$$

6.3.3 Comparação entre os dois modelos

A clara desvantagem do modelo de caminhos em relação ao modelo de arcos é a necessidade de ter uma representação explícita dos caminhos existentes na rede. Note-se que, em geral, o número de caminhos cresce exponencialmente com a dimensão da rede. Esta desvantagem pode ser minorada pela utilização de métodos que gerem o conjunto de caminhos candidatos de forma dinâmica, o que envolve um conhecimento mais avançado de programação linear e inteira e a programação de partes do método (i.e. não se pode utilizar um *solver* para programação inteira).

Por oposição, o modelo de arcos não cresce exponencialmente com a dimensão da rede podendo ser introduzido e resolvido directamente por um *solver* genérico de programação linear (inteira).

Existem vantagens da utilização de modelos de caminhos: restrições específicas (eventualmente não lineares) podem ser incorporadas na construção dos conjuntos candidatos, o modelo tem menos restrições do que o modelo de arcos, e tipicamente é mais fácil de resolver.

6.4 Encaminhamento com restrições adicionais

6.4.1 Encaminhamento por um caminho único

Modelo de arcos

No caso de se pretender que apenas um caminho seja usado por cada comodidade, as restrições do modelo de arcos

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

devem ser substituídas por

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

Modelo de caminhos

No caso de se pretender que apenas um caminho seja usado por cada comodidade, as restrições de domínio do modelo de caminhos devem ser substituídas por

$$x^{pk} \in \{0,1\}, \forall k \in K, \forall p \in P^k.$$

6.4.2 Encaminhamento por um número limitado de caminhos

Modelo de caminhos

O caso geral de limitar o número de caminhos de uma comodidade k a uma constante f^k requer a utilização de variáveis binárias para contarem o número de caminhos que está a ser usado.

Definem-se assim as seguintes variáveis de decisão:

- $w^{pk} = \begin{cases} 1, se\ o\ caminho\ p\ da\ comodidade\ k\ é\ usado \\ 0, caso\ contrário \end{cases} \quad \forall k \in K, \forall p \in P^k$

As restrições

$$x^{pk} \leq w^{pk}, \forall k \in K, \forall p \in P^k$$
$$w_{ij}^k \in \{0,1\}, \forall ij \in A, \forall k \in K$$

asseguram que, sempre que um caminho tenha fluxo não nulo ($x^{pk} > 0$), a variável w^{pk} correspondente tem valor 1.

As restrições

$$\sum_{p \in P^k} w^{pk} \leq f^k, \forall k \in K$$

limitam o número de caminhos para cada comodidade.

Modelo de arcos

No modelo de arcos, dado não existirem variáveis associadas a caminhos, a limitação ao número de caminhos usado não é trivial. A solução apresentada na Figura 6.3. tem fluxo em três caminhos. Se existisse um limite de 2 ao número caminhos, esta solução não seria admissível e é difícil modelar essa situação através de variáveis associadas aos fluxos nos arcos (mesmo adicionando variáveis binárias indicando se há fluxo num arco ou não – note-se que embora haja três caminhos, o número de arcos com fluxo de entrada ou de saída de um nodo nunca excede 2).

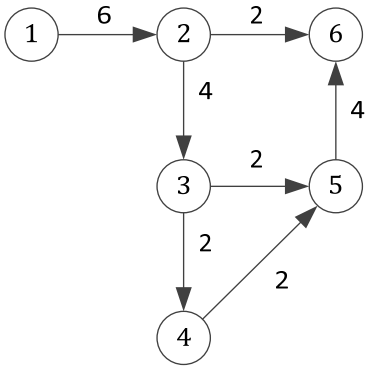


Figura 6.3. Solução de um problema de uma comodidade com origem 1 e destino 6. Junto aos arcos representam-se os fluxos nos arcos da solução.

6.4.3 Diversidade de caminhos

Modelo de caminhos

Em certas situações pode ser desejável que a procura seja dividida por mais de um caminho para que, no caso de uma falha num arco, o fluxo não seja demasiado afectado.

Define-se g^k como sendo o factor de diversidade da comodidade k, i.e. o número mínimo de caminhos que a comodidade k tem de usar.

O seguinte grupo de restrições garante que a fracção do fluxo em cada caminho não excede a proporção máxima para cada caminho.

$$x^{pk} \leq \frac{1}{g^k}, \forall k \in K, \forall p \in P^k$$

Caso a diversidade seja definida nos arcos, estas restrições deverão passar a:

$$a_{ij}^{pk} x^{pk} \leq \frac{1}{g^k}, \forall ij \in A, \forall k \in K, \forall p \in P^k$$

Modelo de arcos

No modelo de arcos a diversidade nos arcos é modelada através de:

$$x_{ij}^k \leq \frac{1}{g^k}, \forall ij \in A, \forall k \in K$$

6.4.4 Fluxo mínimo nos arcos com fluxo

Modelo de arcos

Nalguns problemas pode pretender-se que o fluxo num arco de uma comodidade seja zero ou maior do que um determinado valor (menor que a procura), l_{ij}^k. Para modelar este problema, consideram-se as variáveis de decisão adicionais:

- $y_{ij}^k = \begin{cases} 1, se\ fluxo\ da\ comodidade\ k\ no\ arco\ ij\ é\ maior\ do\ que\ zero \\ 0, caso\ contrário \end{cases} \forall k \in K, \forall ij \in A$

As restrições adicionais são:

$$x_{ij}^k \leq y_{ij}^k, \forall k \in K, \forall ij \in A$$
$$b^k x_{ij}^k \geq l_{ij}^k y_{ij}^k, \forall k \in K, \forall ij \in A$$
$$y_{ij}^k \in \{0,1\}, \forall ij \in A, \forall k \in K$$

Modelo de caminhos

Através de

$$x_{ij}^k = \sum_{p \in P^k} a_{ij}^{pk} x^{pk}, \forall ij \in A, \forall k \in K$$

e usando as mesmas variáveis de decisão adicionais que no modelo de caminhos, modela-se o mesmo problema com caminhos.

6.5 Objectivos alternativos

6.5.1 Carga nas ligações

Modelo de arcos

Por definição, a carga de um arco é a razão entre o fluxo que o atravessa e a sua capacidade. A carga de um arco é assim uma medida da sua (potencial) congestão.

Um objectivo razoável para o problema de fluxo multicomodidade, em alternativa ao problema da minimização do custo total até agora usado, é assim a minimização de uma medida relacionada com a carga.

Definem-se as variáveis de decisão

$$v_{ij} - carga\ do\ arco\ ij, \forall ij \in A$$

Para a minimização da carga média (ou, de forma equivalente, da soma da carga de todos os arcos), incluem-se no modelo de arcos as restrições

$$v_{ij} = \frac{\sum_{k \in K} b^k x_{ij}^k}{u_{ij}}, \forall ij \in A$$

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

A função objectivo é

$$Min\ z = \sum_{ij \in A} v_{ij}$$

Caso se pretenda a minimização da maior carga de um arco é necessária uma nova variável de decisão:

$$v - maior\ carga$$

e as restrições

$$v \geq v_{ij}, \forall ij \in A$$

$$v_{ij} = \frac{\sum_{k \in K} b^k x_{ij}^k}{u_{ij}}, \forall ij \in A$$

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

$$v \geq 0$$

A função objectivo passa a ser

$$Min\ z = v$$

Modelo de caminhos

A minimização da carga pode ser modelada através de caminhos de forma semelhante à modelação por arcos.

6.5.2 Número de saltos de encaminhamento

Modelo de arcos

O número de saltos de um caminho é o número de arcos que o compõem. Um modelo para a minimização do número médio (ou total) de saltos dos caminhos associados às comodidades (no caso de encaminhamento por um caminho para cada comodidade) contém as variáveis de decisão adicionais:

Definem-se as variáveis de decisão

$$t^k - número\ de\ saltos\ do\ caminho\ da\ comodidade\ k, \forall k \in K$$

As restrições seguintes relacionam estas variáveis com as variáveis de fluxo nos arcos.

$$t^k = \sum_{ij \in A} x_{ij}^k, \forall k \in K$$

$$t^k \geq 0, \forall k \in K$$

A função objectivo é

$$Min\ z = \sum_{k \in K} t^k$$

Caso se pretenda a minimização do maior número de saltos é necessária uma nova variável de decisão:

$$t - maior\ número\ de\ saltos$$

e as restrições

$$t \geq t^k, \forall k \in K$$
$$t^k = \sum_{ij \in A} x_{ij}^k, \forall k \in K$$
$$t^k \geq 0, \forall k \in K$$
$$t \geq 0$$

A função objectivo passa a ser

$$Min\ z = t$$

Modelo de caminhos

No modelo de caminhos, é conhecido o número de saltos de cada caminho p de cada comodidade k, que se representa aqui por s^{pk}.

No caso da minimização da média, tem-se a função objectivo

$$Min\ z = \sum_{k \in K} \sum_{p \in P^k} s^{pk} x^{pk}$$

No caso da minimização do maior número de saltos, tem-se a função objectivo

$$Min\ z = t$$

e as restrições

$$t \geq s^{pk} x^{pk}, \forall k \in K, \forall p \in P^k$$
$$t \geq 0$$

6.5.3 Capacidade residual

Modelo de arcos

Neste objectivo pretende-se que a soma das capacidades residuais dos arcos (i.e, a capacidade disponível após efectuado o encaminhamento) seja a maior possível. A função objectivo é assim:

$$Max\ z = \sum_{ij \in A} \left(u_{ij} - \sum_{k \in K} b^k x_{ij}^k \right)$$

ou, de forma equivalente, já que $\sum_{ij \in A} u_{ij}$ é constante,

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

Note-se a relação próxima entre objectivo e o da minimização do número total de saltos.

Modelo de caminhos

A maximização da capacidade residual pode ser modelada através de caminhos de forma semelhante a modelação por arcos.

6.6 Capacidades e custos nos nodos

Até ao momento considerou-se que as capacidades e os custos (quando existentes) estavam associados aos arcos. A modelação de problemas em que estejam associadas aos nodos pode ser feita de duas formas.

A primeira consiste na criação de uma rede modificada em que cada nodo é substituído por dois nodos um de entrada e um de saída, unidos por um arco, como ilustrado na Figura 6.4. Dessa forma o tratamento do fluxo que atravessa o nodo original pode ser feito considerando apenas arcos como anteriormente. Por exemplo, se a capacidade de um nodo i é de u_i, essa capacidade estará associada ao arco $i'i''$.

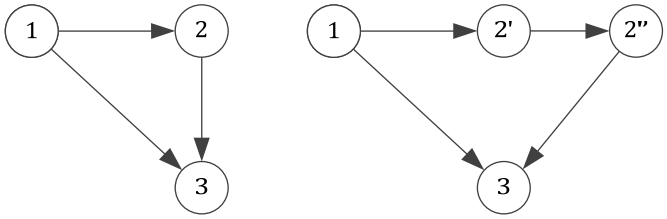


Figura 6.4. O nodo 2 é substituído pelos nodos 2′e 2′′ e a sua capacidade associada ao arco (2′,2′′).

A segunda consiste em representar o fluxo que atravessa um nodo como a soma do fluxo dos arcos que dele saem ou que nele entram (dependendo do caso concreto):

$$x_i = \sum_{j:ij \in A} x_{ij}$$

ou

$$x_i = \sum_{j:ji \in A} x_{ji}$$

e considerando um limite superior correspondente à capacidade:

$$x_i \leq u_i$$

Embora não tenham sido consideradas comodidades, na sua presença, aplica-se o mesmo raciocínio.

6.7 Estruturas de custos

6.7.1 Custo fixo e custo com parcela fixa e parcela linear

Considera-se que existe um custo pela utilização de um arco independente do fluxo que o atravessa. Este variante é particularmente relevante no desenho de redes em que à utilização planeada de um arco corresponde um custo de instalação de equipamento / ligações.

Representa-se o custo fixo do arco ij por $f_{ij}, \forall ij \in A$. Para modelar o custo associado aos fluxos nos arco, são necessárias variáveis de decisão adicionais:

- $y_{ij} = \begin{cases} 1, se\ existe\ fluxo\ no\ arco\ ij \\ 0, caso\ contrário \end{cases}, \forall ij \in A$

O modelo de fluxo multicomodidade passa a

$$Min\ z = \sum_{ij \in A} f_{ij} y_{ij}$$

$$sujeito\ a:$$

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

$$\sum_{k \in K} b^k x_{ij}^k \leq u_{ij} y_{ij}, \forall ij \in A$$

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

$$y_{ij} \in \{0,1\}, \forall ij \in A.$$

Note-se que a restrição de capacidade passa também a relacionar os dois tipos de variáveis de decisão: x_{ij}^k e y_{ij}.

Pode também considerar-se modelo que comporta custos fixos e lineares:

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

$$sujeito\ a:$$

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

$$\sum_{k \in K} b^k x_{ij}^k \leq u_{ij} y_{ij}, \forall ij \in A$$

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

$$y_{ij} \in \{0,1\}, \forall ij \in A.$$

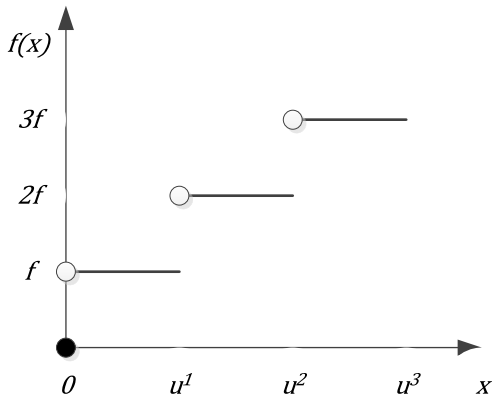
6.7.2 Custo em degrau

Se a função custo é em degrau as restrições do modelo anterior

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

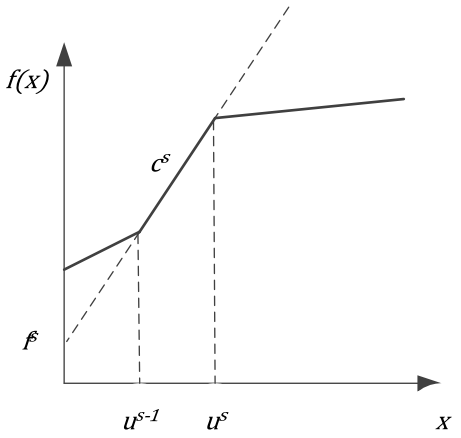
devem ser substituídas por

$$y_{ij} \geq 0\ e\ inteiro, \forall ij \in A$$



6.7.3 Custo linear por partes

A título exemplificativo, apresenta-se uma função linear por partes com três segmentos. Representam-se os parâmetros associados ao segmento central (custo marginal – declive da recta, c^s, custo na origem, f^s, e início e final do segmento, b^{s-1} e b^s).



Definem-se as variáveis de decisão

$$y^s = \begin{cases} 1, se\ fluxo\ se\ situa\ no\ segmento\ s \\ 0, caso\ contrário \end{cases}, \forall s \in S$$

$$z^s - fluxo\ total\ se\ o\ fluxo\ se\ situa\ no\ segmento\ s, \\ ou\ 0\ caso\ contrário\ \forall s \in S$$

O fluxo situa-se apenas num segmento

$$\sum_{s \in S} y^s = 1$$

Verificando-se

$$b^{s-1} y^s \leq z^s \leq b^s y^s, \forall s \in S$$

A restrição

$$x = \sum_{s \in S} z^s$$

relaciona o fluxo total com o fluxo dos segmentos (só um é positivo).

A função objectivo é:

$$min\, z = \sum_{s \in S} (c^s z^s + f^s y^s)$$

Uma estrutura que também é definida por segmentos é a estrutura de custos em degrau em que $c^s = 0, \forall s \in S$ e os segmentos não são necessariamente contínuos.

6.8 Múltiplas origens e múltiplos destinos

Nalguns problemas, uma comodidade não está associada apenas a uma origem e a um destino mas a múltiplas origens e destinos. Assim, para cada comodidade e cada nodo, representa-se a oferta / procura de um nodo i para a comodidade k por e_i^k. Se $e_i^k > 0$, o nodo é uma origem para a comodidade. Se $e_i^k < 0$, o nodo é uma origem para a comodidade. Se $e_i^k = 0$, o nodo não é origem nem destino para a comodidade (é um nodo de passagem ou de transbordo).

As variáveis de decisão são:

- $x_{ij}^k = fluxo\ da\ comodidade\ k\ no\ arco\ ij, \forall ij \in A, \forall k \in K$

O modelo é:

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

$$sujeito\ a:$$

$$\sum_{j:ij \in A} x_{ij}^k - \sum_{i:ji \in A} x_{ij}^k = e_i^k, \forall i \in N,\ k \in K$$

$$\sum_{k \in K} x_{ij}^k \leq u_{ij}, \forall ij \in A$$

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

É de referir que se cada comodidade tem uma origem (destino) e múltiplos destinos (origens), tanto o modelo de origem-destino únicos como o modelo de múltiplas origens destinos podem ser usados (dependendo da definição de comodidade).

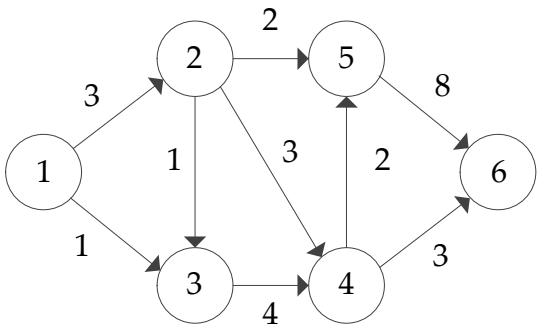
6.9 Exercícios

Exercício 6.1 Custo fixo

Considere a rede da figura e duas comodidades, uma com origem em 1 e destino 4 e outra com origem em 1 e destino 5. Os valores juntos aos arcos correspondem ao custo fixo de se utilizar um arco. Não existem custos proporcionais ao fluxo.

a) Por inspecção, indique uma solução óptima.

b) Apresente um modelo de programação inteira.



Exercício 6.2 Número exacto de caminhos

Modele o problema de o fluxo de uma comodidade k ter de seguir exactamente h^k caminhos com igual fluxo em todos eles.

Exercício 6.3 Capacidades nos nodos

Apresente um modelo de fluxo multicomodidade em que existam capacidades associadas aos nodos, representando a capacidade do nodo i por u_i. Um nodo ter capacidade siginifica que o fluxo que atravessa o nodo não pode exceder um determinado valor.

Exercício 6.4 Rede de telecomunicações

Considere a rede de telecomunicações representada na Figura 6.5. Os arcos representados a carregado têm uma capacidade total, i.e. considerando ambos os sentidos, de 12 MBps. Os restantes arcos têm uma capacidade de 10 MBps. Os principais fluxos de tráfego que se estimam para esta rede são os dados na Tabela 6.1.

Apresente um modelo de programação inteira para o problema de encaminhamento dos quatro fluxos por caminho único para cada um dos objectivos referidos na seccção 6.5.

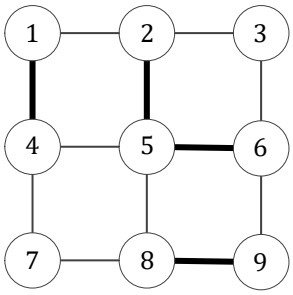


Figura 6.5. Rede.

Índice	Origem	Destino	Tráfego (MBps)
1	1	9	7
2	2	8	6
3	3	7	5
4	9	1	4

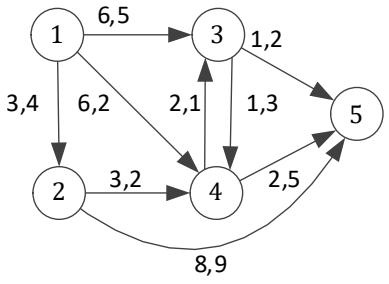
Tabela 6.1. Tráfego.

Exercício 6.5 Minimização da maior carga

Apresente um modelo para minimização da maior carga baseado em caminhos.

Exercício 6.6 Modelos e software

Considere a rede da figura e duas comodidades, a primeira com origem em 1, destino 5 e procura 5, e a segunda com origem em 2, destino 5 e procura 4. Os custos unitários são representados junto a cada arco. A capacidade de cada arco é de 6.



Apresente modelos de programação linear / inteira baseados em arcos e baseados em caminhos e obtenha uma solução óptima através da utilização de software para os seguintes problemas:

a) base;

b) encaminhamento por caminho único;

c) custo fixo de 10 associado à utilização de cada arco;

d) para além do custo fixo referido na alínea c), existe uma limitação ao orçamento de 30.

Exercício 6.7 Dimensionamento de rede

Uma operadora de telecomunicações pretende implementar uma rede, tendo já decidido a localização dos nodos, faltando decidir as ligações a estabelecer entre eles. Tendo em conta que é conhecida a matriz de tráfego, apresente um modelo de programação inteira para os seguintes casos:

a) Ao estabelecimento de uma ligação está associado um custo fixo.

b) O custo de instalação de uma ligação é modular – a cada módulo instalado corresponde um aumento da capacidade e um aumento do custo (por exemplo, cada módulo custa c e tem capacidade de u, logo se forem instalados 5 módulos, a capacidade instalada é de $5u$ e o custo é de $5c$).

c) Existem diferentes tipos de ligações alternativas (diferentes custos e capacidades) para cada ligação.

Exercício 6.8 Rede de computadores

Pretende-se estabelecer três caminhos numa rede de computadores com 6 nodos. Os arcos existentes e a sua capacidade os dados dos caminhos pretendidos são apresentados nas tabelas.

Arco	Capacidade
(1,2)	30
(1,3)	25
(2,4)	25
(2,5)	30
(3,2)	40
(3,4)	50
(3,5)	20
(4,5)	40
(4,6)	40
(5,6)	40

Caminho	Origem	Destino	Procura
1	1	6	20
2	3	6	15
3	1	5	10

Cada comodidade tem de ser encaminhada por um só caminho e assume-se que o custo de uma unidade atravessar um arco é de uma unidade monetária (U.M.).

a) Apresente um modelo de programação inteira que permita a minimização do custo de encaminhar as três comodidades.

b) Adicionalmente, considere que existe um custo de 10 U.M. por cada arco usado (independentemente do fluxo que o atravessa). Apresente um modelo de programação inteira.

c) Adicionalmente, considere que existe um custo de 20 U.M. por cada nodo usado (no qual se incorre se existir qualquer fluxo a passar por ele). Apresente um modelo de programação inteira.

d) Obtenha a solução óptima dos modelos das três alíneas anteriores através da utilização de software.

Exercício 6.9 Fluxos de tráfego

Apresente modelos de programação inteira para os dois seguintes casos:

- fluxos de tráfego bidireccionais simétricos (igual largura de banda nos dois sentidos) – neste caso, os caminhos de encaminhamento nos dois sentidos têm de ser o mesmo;
- fluxos de tráfego bidireccionais assimétrico (diferentes larguras de banda).

Em ambos os casos, considere que a capacidade está associada à aresta.

Exercício 6.10 Produção e transporte

Uma determinada empresa pretende definir os planos de produção e de transporte dos seus três principais artigos. A empresa tem três fábricas, dois armazéns e, para a semana em consideração, dois clientes principais. A encomenda de cada um dos clientes envolve uma quantidade determinada de cada um dos três artigos.

O processo produtivo de todos os artigos envolve três operações, duas delas realizadas por dois tipos de máquinas diferentes e a outra realizada por pessoas. Os dados existentes para a definição do plano de produção são os seguintes:

- tempo unitário de processamento de cada operação em cada artigo;
- tempo de preparação das máquinas para cada operação em cada artigo;

- disponibilidade das máquinas e dos recursos humanos em cada fábrica;
- custo unitário de produção de cada artigo em cada fábrica.

Os tempos de processamento e de preparação são iguais em todas as fábricas.

Depois de produzidos, os artigos são transportados para armazéns e destes para os clientes. Relativamente ao transporte os dados existentes são os seguintes:

- custo fixo de transportar qualquer quantidade (de um ou mais artigos) entre cada fábrica e cada armazém;
- custo fixo de transportar qualquer quantidade (de um ou mais artigos) entre os dois armazéns;
- custo unitário de transportar uma unidade entre cada armazém e cada cliente (igual para os três artigos).

a) Apresente um modelo de programação inteira (mista) para este problema, considerando que as unidades dos artigos em causa são divisíveis.

b) Obtenha uma solução óptima através do *Solver* do *Excel*.

c) Obtenha uma solução óptima através do OPL Studio.

Os parâmetros são dados nas tabelas seguintes.

Tempo de processamento (horas/unidade)

		Artigo		
		P1	P2	P3
Recurso	M1	7	4	1
	M2	4	6	7
	H	6	10	4

Tempo de preparação (horas)

		Artigo		
		P1	P2	P3
Recurso	M1	66	71	64
	M2	79	66	51

Disponibilidade dos recursos (M1 e M2 em horas, H em horas.homem)

		Recurso		
		M1	M2	H
Fábrica	F1	205	219	200
	F2	180	150	198
	F3	194	191	166

Custo de produção unitário (U.M./unidade)

		Artigo		
		P1	P2	P3
Fábrica	F1	1	1	3
	F2	6	1	8
	F3	7	9	10

Custo de unidade extra de recurso (U.M./hora ou U.M./(homem.hora))

Recurso	M1	5
	M2	1
	H	8

Procura (unidades)

		Artigo		
		P1	P2	P3
Cliente	C1	7	9	12
	C2	5	12	14

Custo fixo de transporte entre fábricas e armazéns (U.M.)

		Armazém	
		A1	A2
Fábrica	F1	23	28
	F2	20	27
	F3	20	25

Custo fixo de transporte entre armazéns (U.M.)

		Armazém	
		A1	A2
Armazém	A1	-	11
	A2	13	-

Custo unitário de transporte entre armazéns e clientes (U.M./unidade)

		Cliente	
		C1	C2
Armazém	A1	2	5
	A2	5	3

6.10 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.

• Pióro, M., & Medhi, D. (2004). Routing, flow, and capacity design in communication and computer networks. Elsevier.

6.11 Resultados de aprendizagem

• Formular problemas de optimização como problemas de fluxo multicomodidade (incluindo variantes e extensões).

• Representar problemas de fluxo multicomodidade através de modelos de programação linear / inteira.

• Resolver problemas de fluxo multicomodidade com *software*, nomeadamente o *Solver* e *openSolver* para *Excel* e o *IBM ILOG CPLEX Optimization Studio*.

7 Encaminhamento de veículos

7.1 Introdução

7.1.1 VRP com capacidades

Os problemas de encaminhamento de veículos ("vehicle routing problem - VRP") são dos mais estudados em Investigação Operacional e Optimização.

Na sua versão mais básica, este problema define-se numa rede em que os nodos representam o depósito onde os veículos estão e os clientes que os veículos têm de visitar. Um arco *ij* corresponde ao cliente *j* (ou o depósito) ser visitado por um veículo após o cliente *i* (estando associado a um peso). Cada cliente tem uma procura conhecida e cada veículo tem uma capacidade. Os dois objectivos mais comuns são a minimização da distância total percorrida e a minimização do número de veículos usados.

Na Figura 7.1 apresenta-se parte de uma instância do VRP com sete clientes (o nodo 0 é o depósito) e uma solução que corresponde a usar três veículos que fazem as rotas 0-1-2-4-5-0, 0-6-7-0 e 0-3-0.

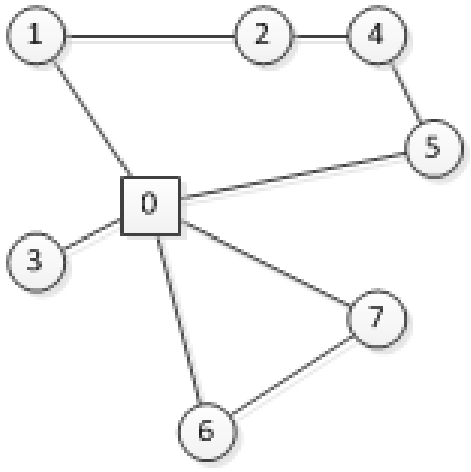


Figura 7.1 Parte de instância do VRP.

O VRP pode ser visto como uma extensão do problema do caixeiro viajante para vários caixeiros viajantes (se as procuras forem 1 e os veículos tiverem capacidade ilimitada, trata-se do "multiple travelling salesman problem") com capacidades e com procuras nos clientes.

Extensões usuais do VRP incluem a consideração janelas temporais nos clientes, tempos de entrega/recolha, veículos com diferentes características e depósitos,

compatibilidade entre clientes e veículos, custos fixos associados à utilização dos veículos. Em termos de objectivos, pode também considerar-se:

- balanceamento de rotas em termos de tempo ou carga dos veículos;
- minimização da soma das penalizações associadas a clientes não servidos (se a procura for maior do que a capacidade dos veículos);
- minimização do custo global de transporte (custo associado às distâncias percorridas e aos veículos usados).

Formaliza-se agora o VRP com capacidades já descrito.

Parâmetros

- $G = (N, A)$ grafo completo;
- $N = \{0, \dots, n\}$, em que 0 representa o depósito;
- K – conjunto de veículos;
- c_{ij} – custo unitário do arco ij, $\forall ij \in A$;
- d_i – procura do cliente i, $i = 1, \dots, n$; $d_0 = 0$;
- w – capacidade de cada veículo.

Se $c_{ij} = c_{ji}$, $\forall ij \in A$, o problema diz-se simétrico, caso contrário diz-se assimétrico.

As seguintes condições são necessárias para o problema se admissível.

- $d_i \leq u, i = 1, \dots, n$, cada encomenda tem de caber num veículo
- $\sum_{i=1}^{n} d_i \leq w|K|$, a capacidade dos veículos tem de ser suficiente para todas as encomendas

Uma solução do VRP corresponde a um conjunto de circuitos elementares (sem subcircuitos e sem repetição de nodos) com as seguintes características: incluem o depósito, cada nodo está associado a um cliente e faz parte de um e só um circuito, a soma da procura dos nodos associados aos clientes de um circuito não excede a capacidade do veículo.

Na maior parte dos problemas, nomeadamente nas definidas no plano, verifica-se a desigualdade triagular: $c_{ik} + c_{kj} \geq c_{ij}$, $\forall i, k, j \in N$. Esta desigualdade representa que o caminho mais curto entre dois nodos é o arco que os une.

7.1.2 Outros VRP

Introduzem-se agora alguns VRP que serão tratados posteriormente.

O problema VRP com janelas temporais incorpora a dimensão tempo no problema anterior. Adicionalmente, consideram-se os seguintes parâmetros:

- $[a_i, b_i]$, cliente i tem de começar a ser servido a partir de a_i e antes de b_i, $i = 1, \dots, n$;
- s_i tempo de serviço no cliente $i = 1, \dots, n$;
- t_{ij} duração do arco $ij, \forall ij \in A$

No VRP com entregas e recolhas (pickup and delivery), cada cliente tem uma encomenda que tem de ser recolhida num nodo e entregue noutro. Neste problema, a carga do veículo ao longo do circuito (quantidade recolhida até ao momento menos a quantidade entregue até ao momento) não pode exceder a capacidade do veículo.

No VRP com capacidades e distâncias, para além da restrição de capacidade dos veículos, cada rota não pode ultrapassar um comprimento (ou tempo) pré-estabelecido (associam-se durações aos arcos e, usualmente, tempos de serviço nos nodos.

No VRP com capacidades e retornos ("backhauls") existem dois tipos de clientes. O primeiro tipo corresponde a clientes onde têm de ser efectuadas entregas. O segundo tipo corresponde a clientes onde têm de ser efectuadas recolhas. Em cada rota, os clientes do primeiro tipo têm de ser todos visitados antes dos clientes do segundo tipo.

7.2 Heurísticas e meta-heurísticas

7.2.1 Heurísticas específicas

Clarke and Wright

Uma das heurísticas mais populares para o VRP é a heurística de Clarke and Wright. A sua diferença em relação à sua versão para o problema do caixeiro viajante é que duas rotas só podem ser fundidas se a procura da rota resultante (soma das procuras das duas rotas fundidas) for menor que a capacidade do veículo.

A heurística baseia-se no cálculo da poupança no caso de duas rotas forem fundidas. Se forem fundidas as duas rotas $(0,\dots,i,0)$ e $(0,j,\dots,0)$ através da remoção dos arcos $i0$ e $0j$ e adição do arco ij, obtém-se a rota $(0,\dots,i,j,\dots,0)$. A poupança obtida é $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ assuminfo que a capacidade do veículo é suficiente para todos os clientes da nova rota.

Os passos da aplicação da heurística de Clarke and Wright são:

- Para cada cliente, criar uma rota $(0,i,0)$, $i = 1,\dots,n$;
- Calcular matriz de poupanças (para todos os pares de nodos, $(n-1)(n-2)$ para instâncias simétricas);
- Ordenar as poupanças por ordem decrescente;
- Enquanto for possível fundir rotas, fundir as duas rotas com maior poupança que resultam numa rota admissível (e.g., a capacidade do veículo ser respeitada);

Notar que a fusão de rotas é sempre por remoção de duas arestas ligadas ao depósito e adição da aresta entre os dois clientes que ficaram com grau 1.

Varrimento

A heurística de varrimento (sweep) é uma heurística de duas fases do tipo agrupar primeiro, encaminhar depois ("cluster-first, route-second") para instâncias planares. Em cada iteração desta heurística considera-se um veículo ao qual são atribuídos clientes por ordem crescente das amplitudes dos ângulos formados pelas rectas definidas pelo depósito e pelo cliente (ver Figura 7.2). Os clientes são atribuídos ao veículo enquanto a sua capacidade for respeitada. O procedimento é repetido enquanto houver nodos não

atribuídos ir para o início. A segunda fase da heurística corresponde a obter a ordem pela qual cada veículo vai visitar os clientes que lhe foram atribuídos, ou seja, a resolver uma instância do problema do caixeiro viajante para cada veículo. A implementação desta heurística é directa se forem usadas coordenadas polares.

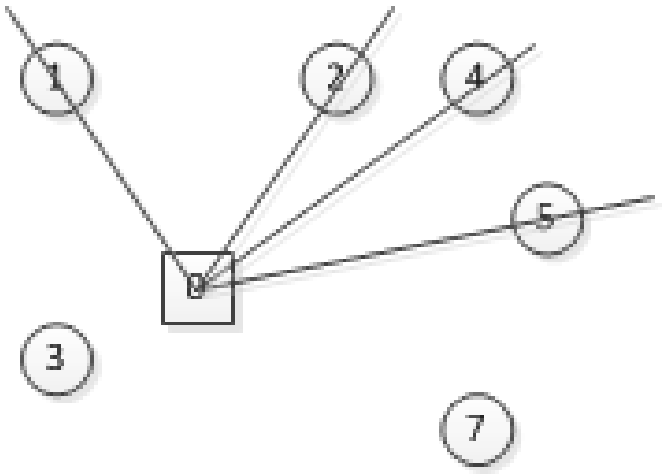


Figura 7.2. Ilustração da primeira fase da heurística de varrimento.

Caixeiro viajante seguido de caminho mais curto

A heurística do caixeiro viajante seguido de caminho mais curto (designação não standard) é uma heurística de duas fases do tipo encaminhar primeiro, agrupar depois ("route-first, cluster-second"). O primeiro passo é obter uma solução para o problema do caixeiro viajante $v(1),\ v(2),\ \ldots,\ v(n)$.

O segundo passo parte da construção de uma rede em que o arco ij corresponde a visitar os clientes de $v(i+1)$ até $v(j)$ numa rota, $i < j$. O comprimento dessa rota tem três parcelas, ir do depósito até ao cliente i, ir do cliente até ao depósito e o comprimento de visitar os clientes entre i e j. Assim, se rota for admissível, $d_{ij} = c_{0(i+1)} + \sum_{k=i+1}^{j-1} c_{k(k+1)} + c_{j0}$. Se não, $d_{ij} = \infty$. O caminho mais curto entre 0 e n na rede construída desta foram indica as rotas. Um nodo i fazer parte do caminho mais curto na rede construída quer dizer que em i+1 começa uma rota que termina em j em que j é o nodo destino do arco que faz parte do caminho e tem origem em i.

No exemplo das Figura 7.3 e Figura 7.4, Se o caminho mais curto na rede inclui os arcos 0-1-3, duas rotas são 0-1-0 e 0-2-3-0.

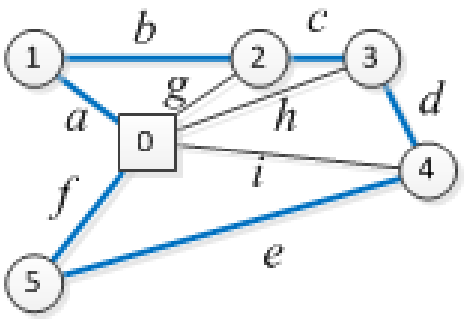


Figura 7.3. Solução da primeira fase.

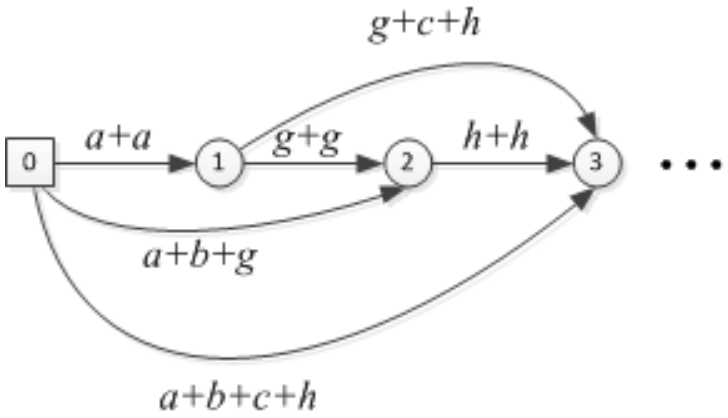


Figura 7.4. Rede da segunda fase.

7.2.2 Heurísticas de pesquisa local

Movimentos envolvendo apenas uma rota

Qualquer movimento de pesquisa local para o problema do caixeiro viajante (2-opt, 3-opt, troca de dois vértives) pode ser utilizado em heurísticas de pesquisa local para o VRP.

Os movimentos seguintes envolvem mais do que uma rota.

Cruzamento

O movimento de cruzamento envolve remover dois arcos de rotas diferentes e voltar a ligá-los de outra forma. Na ilustração da Figura 7.5, os arcos $(i, i+1)$ e $(j, j+1)$ são removidos e o arco (i, j) é inserido numa rota e arco $(i+1, j+1)$ na outra.

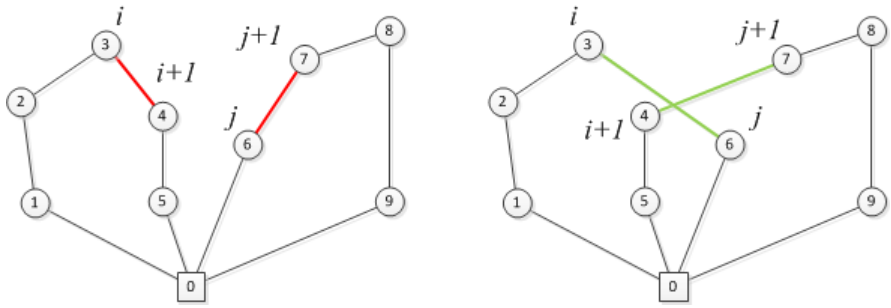


Figura 7.5. Movimento de cruzamento.

Troca

O movimento de troca consiste em trocar dois caminhos (que podem ter apenas um nodo) entre duas rotas. Na ilustração da Figura 7.6, os nodos $i+1$ e $j+1$ são trocados. Tal implica inserir os arcos $(i, j+1)$ e $(i+1, j+2)$ numa rota e arcos $(j, i+1)$ e $(j+1, i+2)$ na outra

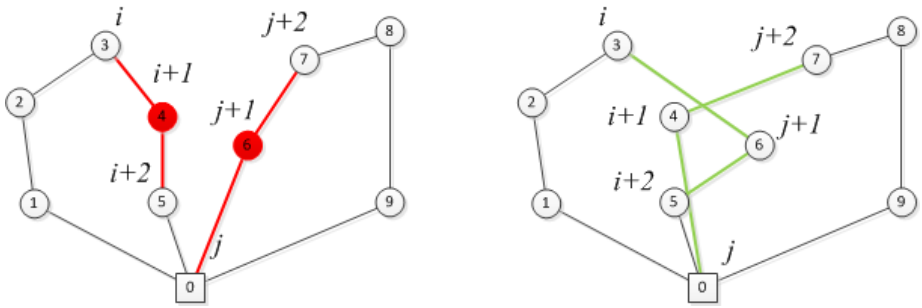


Figura 7.6. Movimento de troca.

Realocação

No movimento de realocação de um caminho, um caminho (que pode ter apenas um nodo) é alocado a uma rota diferente. Na Figura 7.7 ilustra-se a realocação do nodo $i+1$. Os arcos $(i, i+1)$, $(i+1, i+2)$ e $(j, j+1)$ e $(j+1, j+2)$ são removidos e os arcos $(j, i+1)$, $(i+1, j+1)$ e $(i, i+2)$ são inseridos.

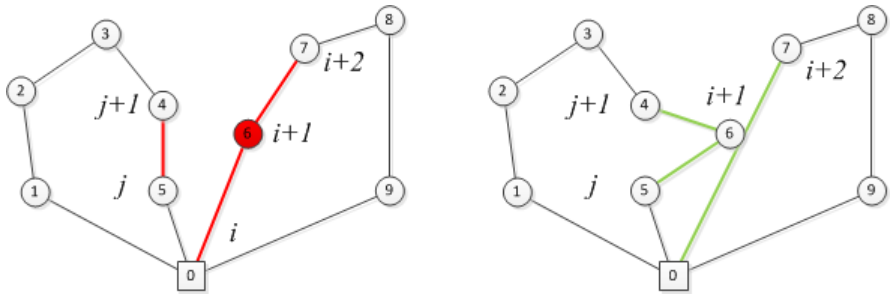


Figura 7.7. Movimento de realocação.

7.3 Programação inteira

Considera-se o problema VRO com capacidades assimétrico.

7.3.1 Modelo de fluxo de dois índices

As variáveis de decisão são

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

O modelo é

$$Min\ z = \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij}$$

$$sujeito\ a:$$

$$\sum_{j \in N} x_{ij} = 1\ , \forall i \in N \setminus \{0\}$$

$$\sum_{j \in N} x_{ji} = 1\ , \forall i \in N \setminus \{0\}$$

$$\sum_{j \in N} x_{0j} = |K|$$

$$Restrições\ de\ eliminação\ de\ subcircuitos$$

$$x_{ij} \in \{0,1\}, \forall i, j \in N$$

O primeiro e segundo grupo de restrições garantem que todos os clientes são visitados. O terceiro grupo de restrições fica o número de veículos que vão ser usados (o número de circuitos).

7.3.2 Modelo de três índices

As variáveis de decisão são

$$x_{ijk} = \begin{cases} 1, se\ arco\ ij\ faz\ parte\ do\ circuito\ k \\ 0,\ caso\ contrário \end{cases}, \forall ij \in A, \forall k \in K$$

$$y_{ik} = \begin{cases} 1, se\ nodo\ i\ faz\ parte\ do\ circuito\ k \\ 0,\ caso\ contrário \end{cases}, \forall i \in N, \forall k \in K$$

O modelo é

$$Min\ z = \sum_{i \in N} \sum_{j \in N} c_{ij} \sum_{k \in K} x_{ijk}$$

$$sujeito\ a:$$

$$\sum_{j \in N} x_{0jk} \leq 1, \forall k \in K$$

$$\sum_{k \in K} y_{ik} = 1\ , \forall i \in N \setminus \{0\}$$

$$\sum_{j \in N} x_{ijk} = y_{ik}, \forall i \in N, \forall k \in K$$

$$\sum_{j \in N} x_{jik} = y_{ik}, \forall i \in N, \forall k \in K$$
$$\sum_{i \in N} d_i y_{ik} \leq w, \forall k \in K$$
$$Eliminação\ de\ subcircuitos$$
$$x_{iik} = 0, \forall i \in N, \forall k \in K$$
$$x_{ijk} \in \{0,1\}, \forall i, j \in N,\ \forall k \in K$$
$$y_{ik} \in \{0,1\}, \forall i \in N,\ \forall k \in K$$

A eliminação de subcircuitos pode ser feita através da generalização das restrições de eliminação de subcircuitos de Dantzig-Fulkerson-Johnson para o problema do caixeiro viajante em que a inclusão do depósito em qualquer rota é feita por serem eliminados subcircuitos que não incluam o nodo 0.

$$\sum_{i \in S} \sum_{j \in S} x_{ijk} \leq |S| - 1, \forall S \subseteq N \setminus \{0\}, |S| \geq 2, \forall k \in K$$

A eliminação de subcircuitos pode também obter-se generalizando as restrições de Miller-Tucker-Zemlin para o problema do caixeiro viajante. Para tal definem-se as variáveis de decisão adicionais

$$u_{ik} - carga\ do\ veículo\ k\ \ depois\ de\ visitar\ o\ cliente\ i$$

e as restrições adicionais

$$u_{jk} \geq u_{ik} + w(x_{ijk} - 1) + d_j, \forall k \in K, \forall i, j \in N \setminus \{0\}, i \neq j$$
$$d_i \leq u_{ik} \leq w, \forall i \in N \setminus \{0\}, \forall k \in K$$

O número destas restrições pode ser reduzido, excluindo do modelo os pares de clientes ij para os quais $d_i + d_j > w$.

Consideram-se agora janelas temporais e tempos de serviço associados a cada cliente. A notação usada é a seguinte:

- $[a_i, b_i]$, cliente i tem de começar a ser servido a partir de a_i e antes de b_i, $i = 1, \dots, n$;
- s_i tempo de serviço no cliente $i = 1, \dots, n$;
- t_{ijk} duração do arco ij quando percorrido pelo veículo k, , $\forall ij \in A$, $\forall k \in K$;

As variáveis de decisão adicionais são

- t_{ik} − instante de tempo em que o veículo k começa o serviço em i, $\forall i \in N, \forall k \in K$

E as restrição adicionais

- $t_{jk} \geq t_{ik} + s_i + t_{ijk} - M(1 - x_{ijk})$, $\forall ij \in A, \mathrm{j} \neq 0, \forall k \in K$
- $a_i \leq t_{ik} \leq b_i, \forall i \in N, \forall k \in K$

Note-se que se, $x_{ijk} = 1$, $t_{jk} \geq t_{ik} + s_i + t_{ijk}$ e se $x_{ijk} = 0$, $t_{jk} \geq 0$ para um valor suficientemente grande de M.

7.3.3 Modelo de partição de conjuntos

Consideram-se os seguinte parâmetros

- C - conjunto de todos os circuitos admissíveis indexado por c
- c_c - custo do circuito $c, c = 1, \ldots, |C|$
- $a_{ic} = \begin{cases} 1, \ se\ circuito\ c\ inclui\ o\ nodo\ i \\ 0, \ caso\ contrário \end{cases}, \forall i \in N, \forall c \in C$

As variáveis de decisão são

- $y_c = \begin{cases} 1, \ se\ circuito\ c\ \text{é}\ seleccionado \\ 0, \ caso\ contrário \end{cases}, \forall c \in C$

A definição das restrições de capacidade, distância, janelas temporais é tratada ao nível da definição de C (só inclui circuitos admissíveis) e não explicitamente no modelo.

Quando o número de circuitos é muito elevado, torna-se necessário recorrer a métodos mais elaborados (geração de colunas ou enumeração parcial).

O modelo é

$$Min\ z = \sum_{c \in C} y_c$$

$$sujeito\ a:$$

$$\sum_{c \in C} a_{ic} y_c = 1\ , \forall i \in N \setminus \{0\}$$

$$\sum_{c \in C} y_c = |K|$$

$$y_c \in \{0,1\}, \forall c \in C$$

Se a desigualdade triangular se verificar, pode considerar-se a restrição $\sum_{c \in C} a_{ic} y_c \geq 1\ , \forall i \in N \setminus \{0\}$ em vez do primeiro grupo de restrições (nunca compensa visitar um cliente duas vezes).

7.4 Recolha e entrega com janelas temporais

Para o problema do VRP com recolha e entrega e janelas temporais define-se uma rede em que Cada nodo corresponde a uma origem ou a um destino de uma e só uma encomenda (logo dois nodos podem ter a mesma localização física). Por exemplo, no caso de duas encomendas terem o mesmo local de origem, são criados dois nodos cada um representando a origem de uma das encomenda. Os nodos da rede, V, são os associados às entregas e recolhas mais os associados às localizações iniciais e finais dos veículos.

A notação é a seguinte.

- R – conjunto de encomendas indexadas por r;
- P – conjunto de nodos que são origem de uma encomenda;
- D – conjunto de nodos que são destino de uma encomenda;
- N – conjunto de nodos que são origem ou destino de uma encomenda, $N = P \cup D$;
- $o(r)$, $d(r)$ origem e destino da encomenda r, $\forall r \in R$;
- b_r - quantidade da encomenda r;
- $s(k)$, $t(k)$ origem e destino do veículo $k, \forall k \in K$;
- $l_{o(r)} = b_r, l_{d(r)} = -b_r$;
- t_{ijk} - duração do arco ij quando percorrido pelo veículo k;
- s_i - tempo de serviço no nodo i;
- $V = N \cup \{s(1), s(2), \dots, s(|K|), t(1), t(2), \dots, t(|K|)\}$.

As variáveis de decisão são

- $x_{ijk} = \begin{cases} 1, se\ arco\ ij\ é\ usado\ pelo\ veículo\ k \\ 0,\ caso\ contrário \end{cases}, \forall ij \in A, \forall k \in K$
- t_{ik} = instante de tempo em que o veículo k começa o serviço em i, $\forall i \in N, \forall k \in K$
- u_{ik} = carga do veículo k depois de servir i, $\forall i \in N, \forall k \in K$

A função objectivo é

- $Min\ z = \sum_{ij \in A} \sum_{k \in K} c_{ijk} x_{ijk}$

Descrevem-se agora as restrições.

O primeiro grupo garante que cada encomenda é recolhida e entregue pelo mesmo veículo

$$\sum_{k \in K} \sum_{j \in N \cup t\{k\}} x_{o(r)jk} = 1\ , \forall i \in P$$

$$\sum_{j:o(r)j \in A} x_{o(r)jk} = \sum_{j:jd(r) \in A} x_{jd(r)k}\ , \forall r \in R, \forall k \in K$$

O segundo grupo de restrições diz respeito às janelas temporais e precedência das recolhas em relação a entregas

- $t_{ik} + s_i + t_{ijk} - t_{jk} \leq (1 - x_{ijk})M,\ \forall ij \in A, \forall k \in K$
- $a_i \leq t_{ik} \leq b_i, \forall i \in N, \forall k \in K$
- $t_{o(r)k} + t_{o(r)d(r)k} \leq t_{d(r)k}, \forall r \in \mathrm{R}$

O terceiro grupo de restrições é o relativo à carga dos veículos

- $u_{ik} + l_j - u_{jk} \leq (1 - x_{ijk})M,\ \forall ij \in A, \forall k \in K$
- $u_{ik} + l_j - u_{jk} \geq (1 - x_{ijk})M,\ \forall ij \in A, \forall k \in K$
- $l_{o(r)} \leq u_{o(r)k} \leq u_k, \forall r \in R, \forall k \in K$
- $0 \leq u_{d(r)k} \leq u_k - l_{o(r)}, \forall r \in R, \forall k \in K$
- $u_{s(k)k} = 0, \forall k \in K$

A conservação de fluxo dos veículos

$$\sum_{j:ij\in A} x_{ijk} - \sum_{j:ji\in A} x_{jik} = \begin{cases} 1, \, se \, i = s(k) \\ -1, \, se \, i = t(k) \\ 0, \, se \, i \neq s(k), i \neq t(k) \end{cases}, \forall k \in K, \, \forall i \in N$$

As restrições de domínio das variáveis de decisão são:

- $x_{ijk} \in \{0,1\}, \forall ij \in A, \forall k \in K$
- $t_{ik} \geq 0, \, \forall i \in N, \forall k \in K$
- $u_{ik} \geq 0, \, \forall i \in N, \forall k \in K$

7.5 Exercícios

Exercício 7.1 Heurísticas

Na tabela apresentam-se as coordenadas no plano de um conjunto de clientes. Existem três veículos para fazer entregas a estes cinco clientes que estão localizados no ponto (0,0), ao qual têm de voltar depois fazer as entregas. Cada veículo não tem capacidade para fazer entregas a mais do que dois clientes.

i	x	y
1	3	8
2	1	4
3	8	2
4	8	4
5	9	3

Obtenha uma solução para o problema de minimizar a distância total percorrida pelos veículos através das heurísticas

a) Clarke and Wright.

b) Varrimento (sweep)-

c) Caixeiro viajante seguido de caminho mais curto.

d) Para uma das soluções obtidas exemplifique um movimento de cruzamento, de troca e de realocação.

Exercício 7.2 Programação inteira

Para o VRP com clientes e distâncias dadas na tabela. Existem três veículos com capacidade 7 e as procuras dos clientes são 3 , 4, 5, 2 e 3 (pela ordem dos seus índices).

a) Apresente um modelo de dois índices.

b) Apresente um modelo de três índices.

c) Apresente um modelo com janelas temporais [0,20] para os clientes 1 e 2 e [10,30] para os restantes clientes.

d) Através da utilização de software, obtenha uma solução óptima para o problema sem janelas temporais.

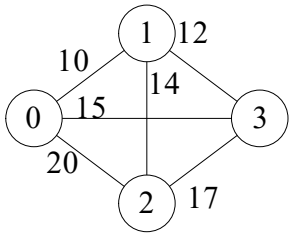
e) Através da utilização de software, obtenha uma solução óptima para o problema com janelas temporais.

	0	1	2	3	4	5
0		3	11	19	13	5
1	18		16	11	13	20
2	3	10		15	16	9
3	4	13	12		6	11
4	7	2	11	16		3
5	2	3	17	6	12	

Exercício 7.3 Partição de conjuntos

Uma determinada empresa de distribuição tem uma frota de veículos para proceder à entrega de encomendas de quatro clientes. A capacidade de cada veículo é de 10 unidades e as encomendas dos clientes são de 5, 4 e 3 unidades. Todos os veículos partem de um mesmo local (depósito), ao qual têm de regressar depois de entregues as encomendas.

Na figura seguinte são dadas distâncias entre todas as localizações relevantes (0 representa a localização inicial e final dos veículos e 1, 2 e 3 representa a localização dos clientes). Pretende-se determinar as rotas a efectuar pelos veículos.



Exercício 7.4 OPL

Apresente o modelo de programação inteira que corresponde ao código OPL abaixo. O modelo diz respeito a que problema?

```
// número de veículos
 int h=...;

// número de clientes
 int n=...;

// capacidade do veículo
 float w=...;

// procura de cada cliente
 float d[1..n]=...;
```

```
// distância entre cada para de clientes (inclui depósito)
 float c[0..n][0..n]=...;

// tempo de serviço
 float s[1..n]=...;

// início da janela temporal
float a[1..n]=...;

// final janela temporal
float b[1..n]=...;

// para exportar para o excel (solução representada a duas dimensões)
int x2d[1..h][0..n];


// variáveis de decisão: x_ij^k
dvar boolean x[1..h][0..n][0..n];

// variáveis de decisão y_i^k
dvar boolean y[1..h][0..n];

// variáveis de decisão
dvar float+ u[1..h][1..n];

execute {
	var custo_max=0;
	for (var i=0; i<n+1;i++) {
		for (var j=0; j<n+1;j++) {
			if (c[i][j]>custo_max) custo_max=c[i][j];
		}
	}
	for (var i=0; i<n+1;i++) {
		if (i==j) c[i][j]=custo_max*n;
	}
	writeln(custo_max*n);
}

minimize sum(k in 1..h) y[k][0];
subject to {
	forall(i in 1..n) sum(k in 1..h) y[k][i]==1;
	forall(i in 1..n, k in 1..h) y[k][i]<=y[k][0];
	forall(i in 0..n, k in 1..h) sum(j in 0..n: i!=j) x[k][i][j]==y[k][i];
	forall(i in 0..n, k in 1..h) sum(j in 0..n: i!=j) x[k][j][i]==y[k][i];
	forall(k in 1..h) sum(j in 1..n) d[j]*y[k][j]<=w;
	forall(i in 1..n, j in 1..n, k in 1..h:i!=j && d[i]+d[j]<w)
u[k][j]>=u[k][i]+w*(x[k][i][j]-1)+d[j];
	forall(k in 1..h, i in 1..n) u[k][i]>=d[i];
	forall(k in 1..h, i in 1..n) u[k][i]<=w;
 }

 execute {
	for (var k=1; k<h+1;k++) {
		for (var i=0; i<n+1;i++) {
			x2d[k][i]=-1;
		}
	}
	for (var k=1; k<h+1;k++) {
		for (var i=0; i<n+1;i++) {
			for (var j=0; j<n+1;j++) {
				if (x[k][i][j]==1) {
					x2d[k][i]=j;
				}
			}
		}
	}

      }
```

7.6 Bibliografia

- Desaulniers, G., Desrosiers, J., Erdmann, A., Solomon, M. M., Soumis, F. (2002). VRP with pickup and delivery. In [TV], chapter 9, 225-242.
- Laporte, G., Semet, F. (2002). Classical heuristics for the capacitated VRP. In [TV], chapter 5, 109-128.
- Networking and Emerging Optimization Research Group, University of Málaga, http://neo.lcc.uma.es/vrp/
- [TV] Toth, P., Vigo, D. (Eds.). (2001). The vehicle routing problem. Siam.
- Zäpfel, G., Braune, R., Bögl, M. (2010). Metaheuristic Search Concepts: A Tutorial with Applications to Production and Logistics. Springer.

7.7 Resultados de aprendizagem

- Aplicar heurísticas específicas para resolver instâncias de pequena dimensão do VRP.
- Exemplificar movimentos de pesquisa local para o VRP.
- Formular o VRP através de modelos de programação inteira.

8 Árvores

8.1 Introdução

Uma árvore num grafo não orientado é um conjunto de arestas ligado que não forma ciclos. Um conjunto de arestas é ligado se existe um caminho entre qualquer par de vértices incidentes nesse conjunto. Uma árvore de suporte é uma árvore que inclui todos os vértices do grafo.

Árvores, e em particular árvores de suporte, são estruturas essenciais em redes. Uma árvore de suporte garante a conectividade de uma rede com o menor número de arestas possível. O problema da árvore de suporte de custo mínimo (ASCM) surge quando existem custos associados à construção/utilização das arestas. Nesse caso, a árvore de suporte de custo mínimo é o conjunto de arcos que garante a conectividade da rede com o menor custo.

Os primeiros algoritmos para problemas de árvores são de meados dos anos 1920.

Árvores de suporte desempenham papel importante noutros problemas (por exemplo, no problema do caminho mais curto entre um nodo e todos os restantes e no problema de fluxo de custo mínimo), sendo também, um ponto de partida de muitos modelos de concepção de redes em que pretende ligar um conjunto de elementos usando o conjunto de estruturas (de ligação ponto a ponto) com menor custo como, por exemplo, em energia, água, redes rodoviárias, e redes de telecomunicações.

O problema da ASCM tem uma aplicação directa na análise de *clusters*. Um *cluster* é um subconjunto de elementos de um conjunto que são próximos (por exemplo, um subconjunto de clientes num problema de distribuição). Adicionar/remover arcos a/de uma ASMC corresponde a juntar/separar *clusters*, como ilustrado na Figura 8.1.

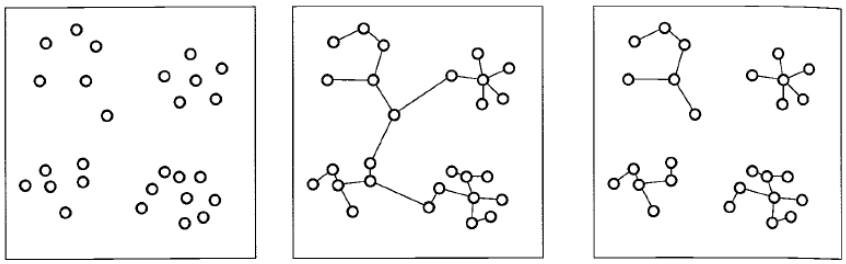


Figura 8.1. Exemplo da utilização de árvores de suporte na definição de *clusters* (Ahuja et al. 1993).

8.2 Condições de optimalidade

8.2.1 Condições de optimalidade baseadas em cortes

Discutem-se agora condições de optimalidade para o problema da ASCM, começando por relembrar a definição de corte. Num grafo $G = (N, A)$, um corte é uma partição do conjunto de vértices N em dois subconjuntos, S e $\bar{S} = N - S$. Cada corte define um conjunto formado pelas arestas que têm uma extremidade em S e outra extremidade em $\bar{S}$.

Uma árvore de suporte T^* é uma ASCM se e só para qualquer aresta $\{i, j\}$ da árvore T^* e para qualquer aresta $\{k, l\}$ do corte que se obtém ao remover $\{i, j\}$ de T^*, se verificar $c_{ij} \leq c_{kl}$.

Considere-se o exemplo da instância da Figura 8.2. Se a aresta {1,4} for removida da árvore, as arestas {1,2}, {3,4} e {3,5} definem o corte obtido e todas têm um custo não inferior a {1,4}. Como o mesmo se verifica para as outras arestas da árvore de suporte ({1,3}, {2,4} e {4,5}), $T* = \{\{1,4\}, \{1,3\}, \{2,4\}, \{4,5\}\}$, é uma ASCM.

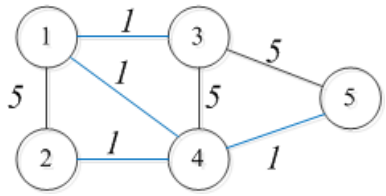


Figura 8.2. Exemplo de ASCM.

8.2.2 Condições de optimalidade baseadas em caminhos

Uma árvore de suporte T^* é uma ASCM se e só se para qualquer aresta $\{k, l\}$ não pertencente a T^* e para qualquer aresta $\{i, j\}$ do caminho que liga k a l em T^*, se verificar $c_{ij} \leq c_{kl}$

No exemplo da Figura 8.2, {3,4} não pertence à árvore, as arestas do caminho de 3 a 4 que pertencem à árvore são as arestas {3,1} e {1,4} e ambas têm um custo não superior a {3,4}. Como o mesmo também se verifica para as outras arestas que não fazem parte da árvore de suporte ({1,2} e {3,5}), $T^* = \{\{1,4\}, \{1,3\}, \{2,4\}, \{4,5\}\}$ é uma ASCM.

8.3 Algoritmos específicos

8.3.1 Kruskal

O algoritmo de Kruskal baseia-se na definição de árvore (de uma rede com n vértices) como um conjunto de $n - 1$ arestas sem ciclos. O algoritmo de Kruskal consiste

em ordenar as arestas por ordem crescente de custo e, considerando, uma aresta de cada vez, testar se a sua inclusão na árvore forma um ciclo. Se a inclusão da aresta não formar um ciclo, é incluída na árvore de suporte, se formar um ciclo passa-se para a aresta seguinte.

O algoritmo é o seguinte.

```
Algoritmo de Kruskal para o problema da árvore de suporte de custo
          mínimo
Entrada:
     rede não orientada com n vértices
     distância associada a cada aresta ij, d_ij, i = 1,...,n, j = 1,...,n, i ≠ j
Saída:
     conjunto de arestas x da árvore de suporte de custo mínimo
// Inicialização
          x = ∅
// C é o conjunto de arestas já examinados
          C = ∅
// Ciclo principal
Enquanto |x| ≠ n − 1
     Repetir
          Seleccionar a aresta ij ∉ C com menor d_ij
          Se ij ∪ x forma ciclo, C = C ∪ ij, aresta_encontrada = falso
          Se não, x = x ∪ ij e aresta_encontrada = verdadeiro
     Enquanto aresta_encontrada = falso
```

Em termos de implementação, o algoritmo de Kruskal começa com uma floresta em que cada árvore tem um vértice. De cada vez que uma aresta é seleccionada, se as suas extremidades estão em árvores diferentes, não forma ciclo e as duas árvores são fundidas, se as suas extremidades estão na mesma árvore, a aresta forma um ciclo. É suficiente para a implementação da detecção de ciclos guardar os vértices de cada árvore e ter um procedimento de fusão de árvores.

O algoritmo de Kruskal, embora seja um algoritmo construtivo e guloso, ao contrário da vasta maioria desses algoritmos, conduz sempre a uma solução óptima. Um algoritmo deste tipo (construtivo e guloso), em cada passo, acrescenta um componente (na ASCM, uma aresta) à solução tendo em conta as consequências imediatas (na ASCM, menor custo) mas sem ter em conta as consequências globais

A solução óptima de uma sua aplicação (Tabela 8.1) na instância da Figura 8.3 é dada na Figura 8.4.

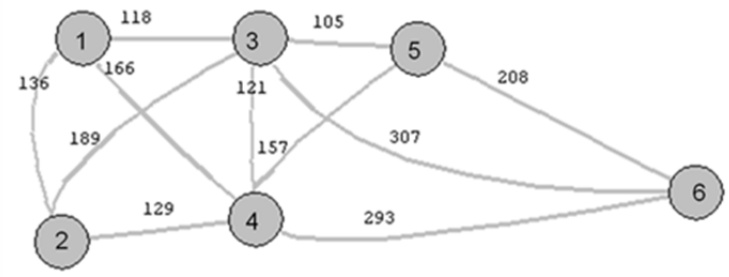


Figura 8.3. Instância do problema da árvore de suporte de custo mínimo.

Iteração	Arco	Custo	Resultado do teste
1	35	105	Incluído na ASCM
2	13	118	Incluído na ASCM
3	34	121	Incluído na ASCM
4	24	129	Incluído na ASCM
5	12	136	Excluído (ciclo 21342)
6	45	157	Excluído (ciclo 3453)
7	14	166	Excluído (ciclo 1341)
8	23	189	Excluído (ciclo 2342)
9	56	208	Incluído na ASCM

Tabela 8.1. Aplicação do algoritmo de Kruskal ao exemplo.

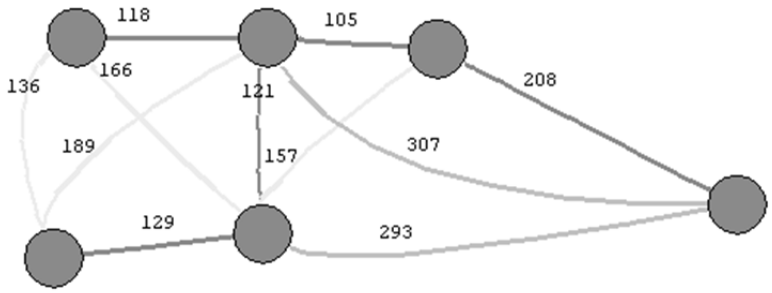


Figura 8.4. Solução óptima (cinco arcos mais escuros) do exemplo da árvore de suporte de custo mínimo.

8.3.2 Prim

O algoritmo de Prim é também um algoritmo guloso que garante a obtenção de uma solução óptima, baseando-se nas condições de optimalidade de cortes. Em cada iteração, é escolhida a aresta de menor custo das que fazem parte do corte definido pela árvore actual.

O algoritmo é o seguinte.

Algoritmo de Prim para o problema da árvore de suporte de custo mínimo
Entrada:
    rede não orientada com n vértices
    distância associada a cada aresta ij, $d_{ij}, i = 1, \dots, n, j = 1, \dots, n, i \neq j$
Saída:
    conjunto de arestas x da árvore de suporte de custo mínimo
// Inicialização
    $x = \emptyset$
// C é o conjunto de vértices que fazem parte da árvore de suporte de custo mínimo
    $C = \{1\}$
// Ciclo principal
Enquanto $|C| \neq n$
    Seleccionar a aresta ij do corte $[C, \bar{C}]$ com menor d_{ij} (em que $i \in C$ e $j \in \bar{C}$)
    $C = C \cup j$
    $x = x \cup ij$

Na Figura 8.5 apresenta-se um exemplo da aplicação do algoritmo de Prim.

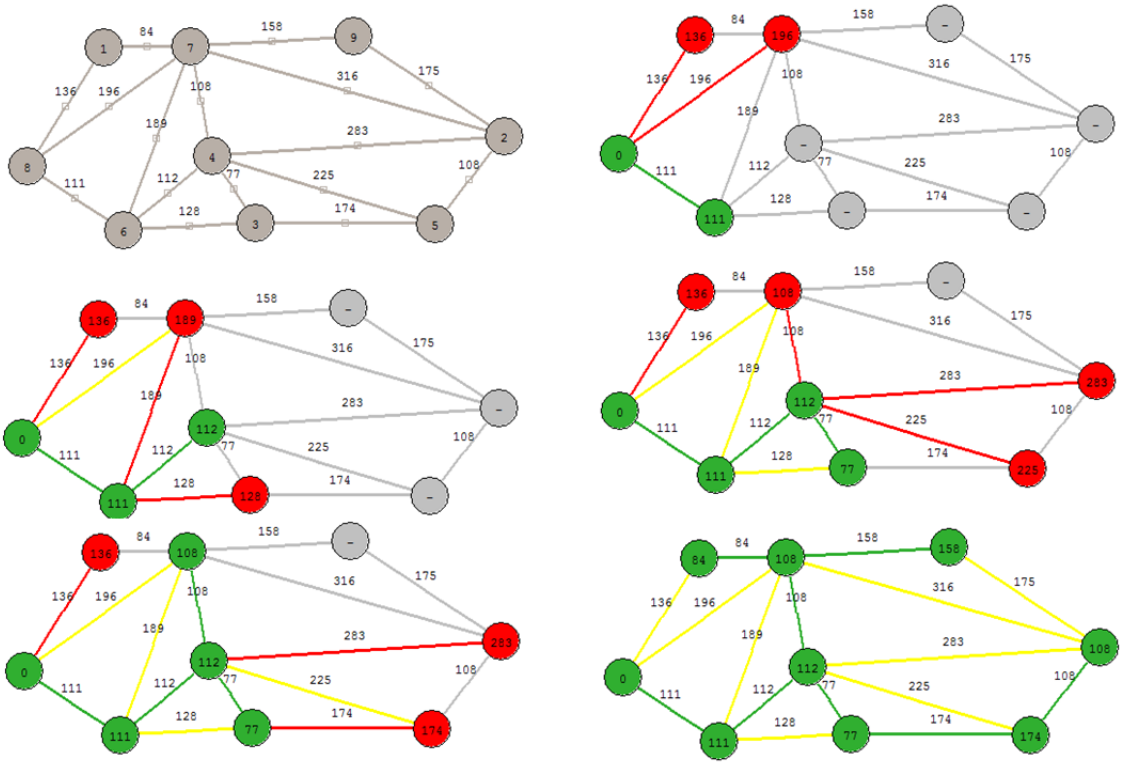


Figura 8.5. Ilustração do algoritmo de Prim.

8.4 Programação inteira

8.4.1 Modelo de cortes

Define-se a seguinte notação e parâmetros:

- $G = (N, A)$ grafo não orientado
- c_{ij} custo unitário da aresta $ij, \forall ij \in A, i < j$
- S conjunto de arestas
- $A(S)$ é o conjunto das arestas com ambas as extremidades no conjunto de vértices S
- $|S|$ número de elementos do conjunto S

Por exemplo, para o grafo da Figura 8.6, se $S = \{1,2,3,4\},\ A(S) = \{12,13,23,34\}$.

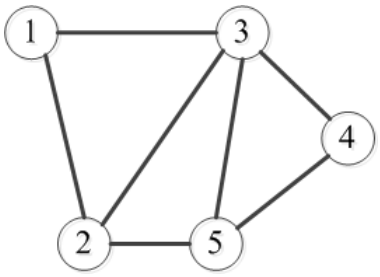


Figura 8.6. Grafo para exemplificação do modelo de cortes.

As variáveis de decisão são

$$x_{ij} = \begin{cases} 1, se\ aresta\ ij\ faz\ parte\ da\ ASCM \\ 0,\ caso\ contrário \end{cases}, \forall ij \in A$$

O modelo é

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

$$sujeito\ a:$$

$$\sum_{ij \in A} x_{ij} = n - 1$$

$$\sum_{ij \in A(S)} x_{ij} \leq |S| - 1, \forall S \subset N$$

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

As restrições de domínio ($x_{ij} \in \{0,1\}, \forall ij \in A$) podem ser substituídas por $0 \leq x_{ij} \leq 1, \forall ij \in A$, já que mesmo efectuando essa substituição existe sempre uma solução óptima em que, para todas as arestas ij, $x_{ij} = 0$ ou $x_{ij} = 1$.

A desvantagem deste modelo é que o número de restrições cresce exponencialmente em função do número de vértices da rede. Uma possível estratégia de resolução consiste em resolver o problema sem o segundo grupo de restrições e, enquanto a solução não for admissível, repetir os seguintes passos: i) identificar um

conjunto de vértices S a que corresponda uma restrição violada, ii) adicionar essa restrição ao problema, iii) resolver problema obtido.

8.4.2 Modelo de fluxos

No modelo de fluxos, substitui-se cada aresta $\{i,j\}$ por dois arcos (i,j) e (j,i). Na rede orientada assim obtida define-se um vértice raiz arbitrário r. O modelo baseia-se em forçar a existência de um caminho entre cada vértice e a raiz r. Se um arco for seleccionado para um caminho então a aresta correspondente é seleccionada para a árvore de suporte de custo mínimo, como exemplificado na

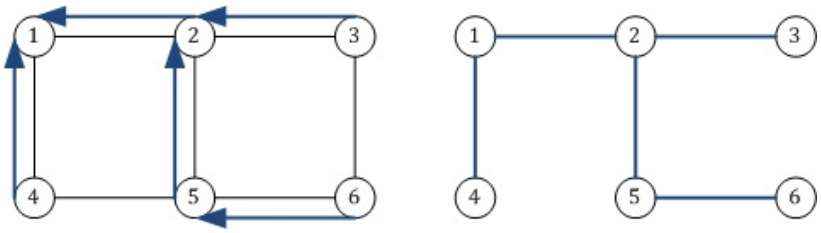


Figura 8.7. Exemplo de transformação de um conjunto de caminhos numa árvore.

Note-se que os caminhos obtidos não correspondem a uma árvore de caminhos mais curtos porque, na ASCM, o custo de um arco só conta uma vez como ilustrado nas Figura 8.8 e Figura 8.9.

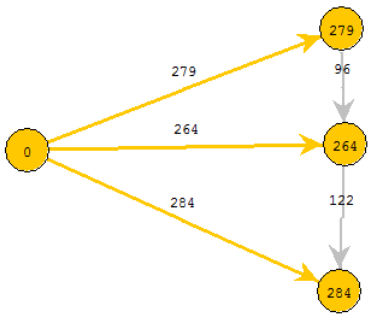


Figura 8.8. Árvore dos caminhos mais curtos.

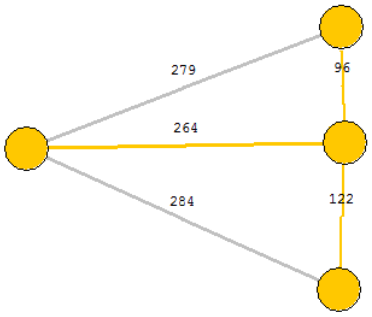


Figura 8.9. Árvore de suporte de custo mínimo.

As variáveis de decisão são

$$x_{ij} = \begin{cases} 1, se\ aresta\ ij\ faz\ parte\ da\ ASCM \\ 0,\ caso\ contrário \end{cases}, \forall ij \in A, i < j$$

$$y_{ijk} = \begin{cases} 1, se\ o\ arco\ ij\ faz\ parte\ do\ caminho\ entre\ k\ e\ a\ raiz\ r \\ 0,\ caso\ contrário \end{cases}, \forall ij \in A$$

O modelo é

$$Min\ z = \sum_{ij \in A, i<j} c_{ij} x_{ij}$$

$$sujeito\ a:$$

$$\sum_{j:ij \in A'} y_{ijk} - \sum_{j:ji \in A'} y_{jik} = \begin{cases} 1,\ se\ i = k \\ 0,\ se\ i \neq k, i \neq r \\ -1, se\ i = r \end{cases}, \forall i \in N, \forall k \in N \backslash \{r\}$$

$$y_{ijk} + y_{jik} \leq x_{ij}, \forall ij \in A$$

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

$$y_{ijk} \in \{0,1\}, \forall ij \in A, \forall k \in N \setminus \{r\}$$

8.5 Extensões

8.5.1 ASCM com restrições de grau

No problema da ASCM com restrições de grau, nenhum vértice da árvore de suporte pode ter mais de b arcos incidentes. Por exemplo, a árvore de suporte da Figura 8.10 não é admissível para $b = 3$ porque o vértice 5 tem grau 4.

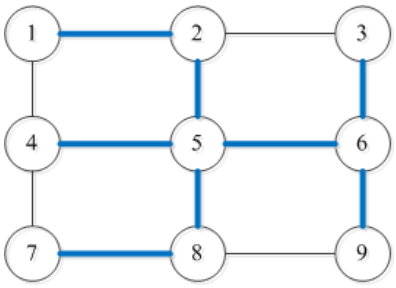


Figura 8.10. Árvore de suporte.

Em programação inteira, as restrições de grau são modeladas por

$$\sum_{ij \in A} x_{ij} \leq b\ , \forall i \in N$$

8.5.2 ASCM com capacidades

No problema da ASCM com capacidades existe um vértice raiz r e os restantes vértices têm procuras de $q_i, \forall i \in N\{r\}$. Neste problema, nenhuma subárvore incidente na raiz r pode ter uma procura total superior a uma dada capacidade Q.

Define-se subárvore principal como uma subárvore que está ligada à raiz apenas por uma aresta e subárvore de um vértice como a subárvore que o contém.

A porta de um vértice j é a aresta que liga a sua subárvore à raiz, representando-se o custo da porta de j por g_j. Define-se c_{ij*} como o custo do arco de menor custo que liga a subárvore i à subárvore j.

No exemplo da Figura 8.11, 5 é a raiz e a solução tem quatro subárvores

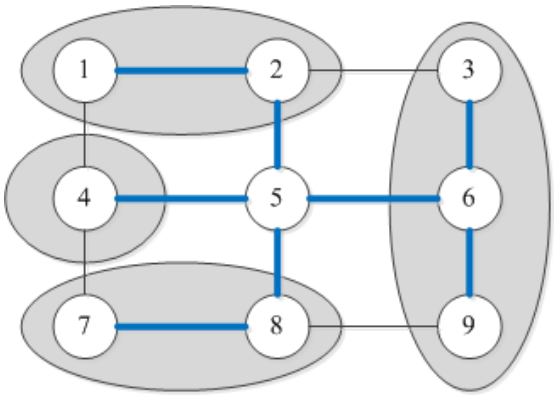


Figura 8.11. Exemplo de subárvores.

Uma heurística para o problema da ASCM com restrições de capacidade é a heurística de Esau-Williams. Nesta heurística, inicializase a solução com uma subárvore principal para cada vértice. Em sguida, para todos os pares de subárvores ij, calcula-se a poupança que se obtém se forem fundidas numa só da forma de menor de custo

$$s_{ij} = \begin{cases} max\{g_i, g_j\} - c_{ij*}, & \text{se a ligação entre as duas subárvores é admissível} \\ -\infty, & \text{se a ligação entre as subárvores não é admissível} \end{cases}$$

A ligação entre duas subárvores não é admissível se não existir nenhum arco entre elas ou se a soma da procura de ambas exceder a capacidade.

Se a maior poupança é positiva, fundem-se as duas árvores associadas a essa poupança e volta-se ao passo de cálculo das poupanças. Se a maior poupança é zero ou negativa, o algoritmo termina.

Apresenta-se de seguida, Figura 8.12 e Figura 8.13, um exemplo da aplicação da heurística de Esau-Williams em que a raiz é o vértice 0.

Q=60					
c_ij	1	2	3	4	5
0	7	9	5	8	8
1		1	4	1	10
2			2	3	4
3				3	8
4					1
q_i	23	34	41	28	15

Figura 8.12. Instância do problema da ASCM com capacidades.

Q=60					
c_ij	1	2	3	4	5
0	7	9	5	8	8
1		1	4	1	10
2			2	3	4
3				3	8
4					1
q_i	23	34	41	28	15

s_ij	1	2	3	4	5
1		**=9-1=8**	-	=8-1=7	=8-10=-2
2			-	-	=9-4=5
3				-	=8-8=0
4					=8-1=7

c_ij	1,2	3	4	5
0	7	5	8	8
1,2		2	1	4
3			3	8
4				1
q_i	57	41	28	15

s_ij	1,2	3	4	5
1,2		-	-	-
3			-	=8-8=0
4				**=8-1=7**

c_ij	1,2	3	4,5
0	7	5	8
1,2		2	1
3			3
4,5			
q_i	57	41	43

s_ij	1,2	3	4,5
1,2		-	-
3			-
4,5			

Figura 8.13. Aplicação da heurística de Esau-Williams.

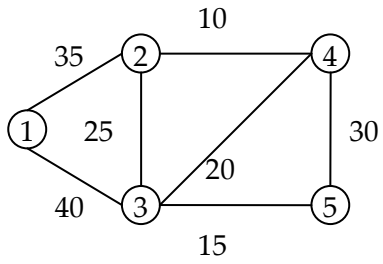
A solução (não necessariamente óptima por se tratar de uma heurística!) obtida é árvore de suporte formada pelas subárvores {01,12}, {03} e {04,45}.

8.6 Exercícios

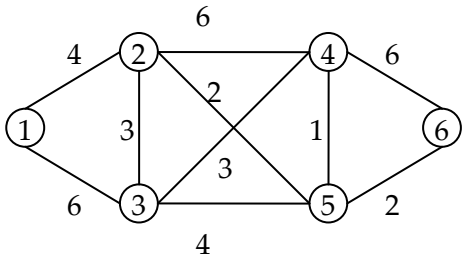
Exercício 8.1 Algoritmos

Utilizando o algoritmo de Kruskal, determine uma árvore de suporte de custo mínimo das seguintes redes. Confirme o resultado que obteve por aplicação do algoritmo de Prim.

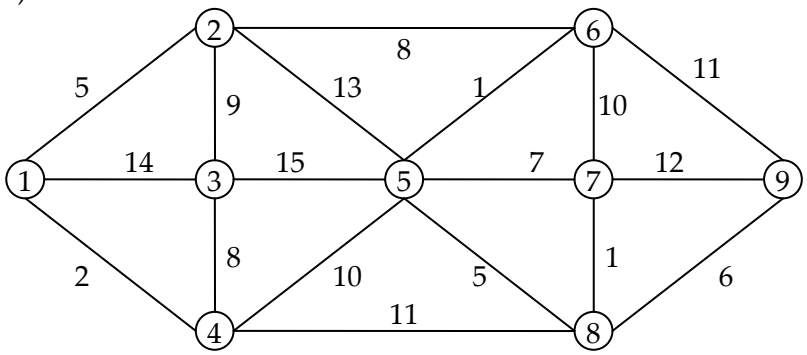
a)



b)



c)



Exercício 8.2 Fibra óptica

Uma empresa de telecomunicações pretende instalar uma rede de cabos de fibra óptica numa região. Na tabela são dados os custos de instalação de um cabo entre cada par de nodos. Pretende-se que todos os nodos possam comunicar entre si (eventualmente utilizando nodos intermédios) com o menor custo possível.

a) Mostre que se trata de um problema de ASCM.

b) Aplique-o os algoritmos de Kruskal e de Prim na obtenção de uma solução.

c) Em que circunstâncias os algoritmos de Kruskal e de Prim pode dar soluções diferentes?

	A	B	C	D	E	F	G
A	–	50	80	40	100	90	20
B		–	120	120	110	90	100
C			–	90	30	90	150
D				–	120	20	100
E					–	100	130
F						–	10
G							–

Exercício 8.3 *Campus* Universitário

Um determinado *Campus* Universitário é constituído por sete edifícios entre os quais existem diversos caminhos pedonais. Tendo em vista ser possível ir de um

qualquer edifício para qualquer outro sempre por caminhos cobertos, foi tomada a decisão de construir coberturas em alguns desses caminhos.

Dados os elevados custos de construção, foi estipulado que o comprimento total de caminho coberto deveria ser o menor possível.

Na tabela seguinte são dados os comprimentos (em metros) dos diversos caminhos existentes entre os vários edifícios. O sinal "–" indica que não há um caminho directo entre os dois edifícios em causa. Por exemplo, o comprimento do caminho entre o edifício A e o edifício B é de 100 metros.

	A	B	C	D	E	F	G
A	–	100	200	300	–	–	–
B	100	–	160	–	210	–	–
C	200	160	–	90	150	120	–
D	300	–	90	–	–	130	–
E	–	210	150	–	–	–	350
F	–	–	120	130	–	–	100
G	–	–	–	–	350	100	–

Quais os caminhos que devem ser cobertos e qual o comprimento total?

Exercício 8.4 Rede de computadores

Na instalação de uma rede com sete computadores pretende-se gastar o menor comprimento de cabo que seja possível, assegurando que todos os computadores podem comunicar entre eles. As distâncias (em metros) entre os locais onde estão os computadores são as dadas na tabela. O sinal "–" indica que não é possível a ligação directa entre os dois computadores em causa. Por exemplo, o comprimento do cabo entre o computador A e o computador B é de 60 metros.

	A	B	C	D	E	F	G
A	–	60	160	260	–	–	–
B	60	–	120	–	170	–	–
C	160	120	–	50	110	80	–
D	260	–	50	–	–	90	–
E	–	170	110	–	–	–	310
F	–	–	80	90	–	–	60
G	–	–	–	–	310	60	–

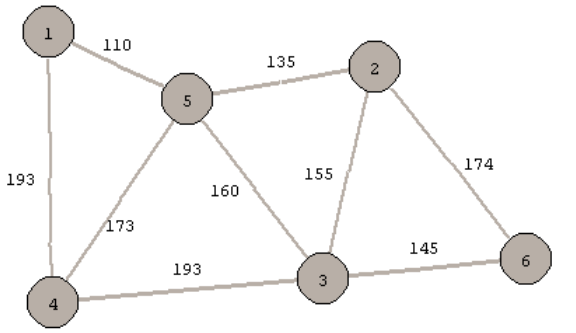
a) Utilize o algoritmo de Kruskal para obter uma solução óptima para este problema.

b) Confirme que a solução é efectivamente óptima com o algoritmo de Prim.

Exercício 8.5 Companhia aérea

Uma companhia aérea pondera a alteração da sua rede de ligações aéreas entre os seis principais aeroportos onde opera. Na rede da figura abaixo indicam-se os custos associados ao estabelecimento das ligações directas entre esses aeroportos (para os casos em que tal é viável). Uma solução interessante para a companhia aérea é estabelecer as ligações directas que conduzem ao menor custo total, permitindo que seja possível ir de qualquer aeroporto para qualquer outro.

Qual é essa solução? Aplique um algoritmo adequado, indicando os passos que efectuar.



Exercício 8.6 Rede de computadores

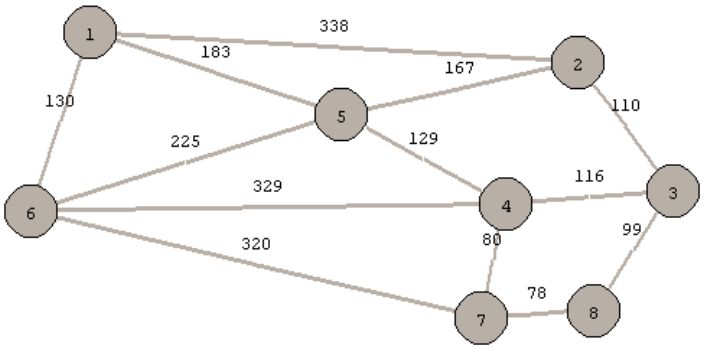
Na figura seguinte representa-se um conjunto de oito computadores existentes num edifício e o conjunto das 13 ligações que podem ser estabelecidas entre eles. Junto a cada potencial ligação é indicada a distância entre os correspondentes dois computadores (em decímetros).

Pretende-se que os computadores funcionem em rede para que seja possível a cada computador comunicar com qualquer outro.

a) Qual o menor número de ligações que é necessário estabelecer?

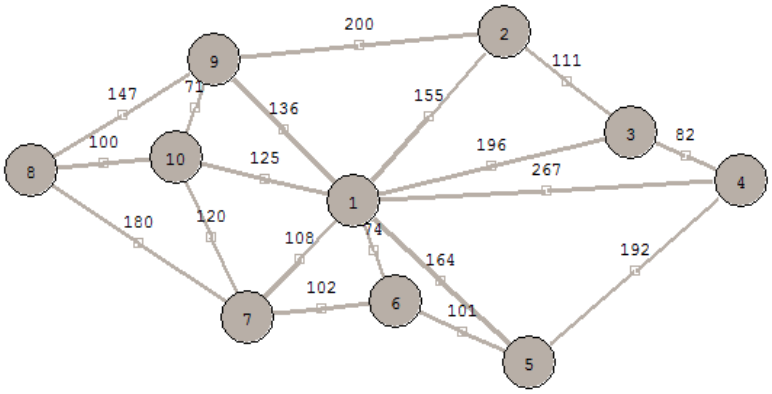
b) Considere que se pretende usar a menor quantidade possível de cabo. Usando um algoritmo adequado e descrevendo claramente a sua aplicação, obtenha uma solução para o problema e indique o seu valor.

c) A solução que obteve na alínea anterior é óptima?



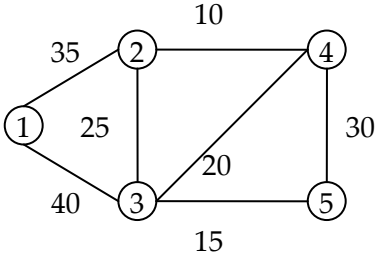
Exercício 8.7 Definição de clusters

Na rede da figura representa-se a localização dos 10 principais clientes de uma empresa de distribuição, bem como as distâncias para os pares de clientes em que há uma ligação directa. A empresa pretende agrupar os clientes em três conjuntos. Defina esses três conjuntos, com base no conceito de árvore de suporte de custo mínimo e num algoritmo adequado.



Exercício 8.8 Programação Inteira

Considere a rede da figura seguinte e o problema da árvore de suporte de custo mínimo.



a) Apresente o modelo de programação inteira de cortes.
b) Obtenha uma solução óptima através da utilização de *software*.

c) Apresente o modelo de programação inteira de fluxos.

d) Obtenha uma solução óptima através com o solver do excel.

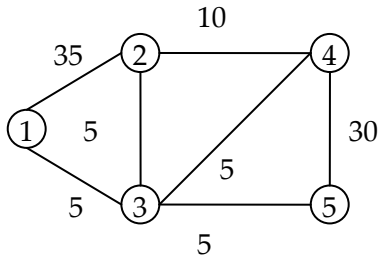
e) Obtenha uma solução óptima através do IBM ILOG CPLEX Optimization Studio.

Exercício 8.9 ASCM com restrição de grau

Considere a rede da figura seguinte e o problema da árvore de suporte de custo mínimo com restrição de grau 3.

a) Apresente um modelo de programação inteira baseado em fluxos.

b) Obtenha uma solução óptima através de um modelo de programação inteira baseado em cortes: i) com o solver do excel, ii) com o IBM ILOG CPLEX Optimization Studio.



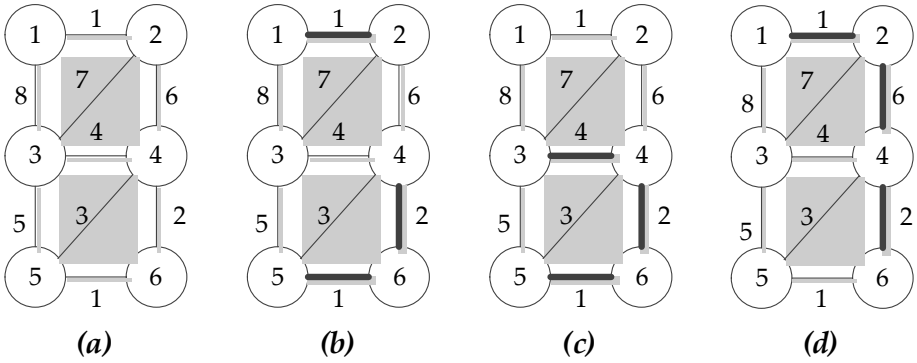
Exercício 8.10 Algoritmos + clusters

Na figura ***(a)*** apresenta-se uma instância do problema da árvore de suporte de custo mínimo. Nas restantes três figuras apresentam-se soluções parciais.

a) Qual ou quais podem ter sido obtidas pelo algoritmo de Kruskal? Justifique.

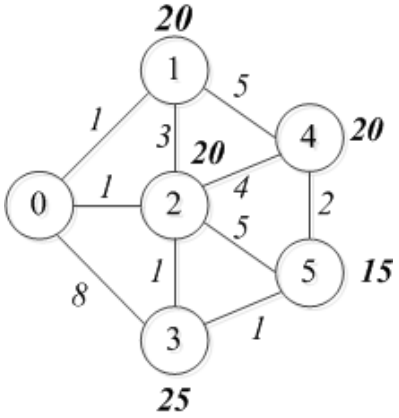
b) Qual ou quais podem ter sido obtidas pelo algoritmo de Prim? Justifique.

c) Considere que os nodos representam clientes de um problema de encaminhamento de veículos com dois veículos. Para resolver o problema, definiu-se o seguinte método: numa primeira fase agrupam-se os clientes e, numa segunda fase, define-se a sequência pela qual são visitados os clientes de cada grupo. Indique uma solução para a primeira fase tendo em conta que a qualidade da solução é medida pela soma do custo de cada grupo. O custo de cada grupo é o custo de uma sua árvore de suporte de custo mínimo (ou seja é a soma dos custos de um subconjunto de arestas que garante a ligação entre qualquer par de clientes do grupo com o menor custo possível).



Exercício 8.11 ASCM com capacidades – Esau-Williams

Obtenha uma árvore de suporte para a rede da figura considerando que a raiz é o vértice 0 e a capacidade é de 60.



Exercício 8.12 ASCM com capacidades – Programação inteira

Apresente um modelo de programação inteira para o problema da ASCM com capacidades.

8.7 Resultados de aprendizagem

- Aplicar o algoritmo de Kruskal resolver instâncias de pequena dimensão do problema da ASCM.
- Aplicar o algoritmo de Prim resolver instâncias de pequena dimensão do problema da ASCM.
- Formular o prolema da ASCM através de modelos de programação inteira.

9 Afectação

9.1 Introdução

O problema de afectação consiste em, dados dois conjuntos e um custo de associação (afectação) para cada par de elementos um de cada conjunto, afectar cada elemento de um conjunto a um elemento do outro conjunto de forma a que o custo total da afectação seja o menor possível.

O primeiro algoritmo eficiente para este problema deve-se a Kuhn em 1955 e teve um importante papel no desenvolvimento da programação linear e da optimização combinatória. Uma revisão do problema de afectação e extensões é feita em (Pentico 2007).

9.2 Programação Inteira

9.2.1 Afectação

Introduz-se o problema de afectação através de um exemplo.

Exemplo 9.1 Problema dos turnos dos enfermeiros

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 9.1 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	7	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	9	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 9.1. 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.

As variáveis de decisão são

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

O modelo de programação inteira é

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

$$sujeito\ a:$$

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

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

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

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.

O problema de afectação pode ser representado numa rede como a da Figura 9.1, onde as arestas correspondem às afectações possíveis.

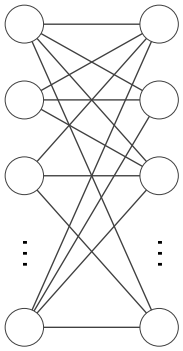


Figura 9.1. Grafo bipartido que representa um problema afectação.

Naturalmente, o número de agentes e tarefas pode ser diferente, acomodando-se essa situação com a alteração do sinal das restrições ou a inserção de agentes e tarefas adicionais (fictícios).

Afectações impossíveis podem ser acomodadas pela remoção dos arcos da rede, forçando as variáveis correspondentes a terem o valor zero ou atribuindo-lhes um custo muito mais elevado do que aos restantes arcos (usualmente representado por M).

9.2.2 Afectação min max

No problema de afectação min max, pretende-se que a afectação com maior valor tenha o menor valor possível. Por exemplo, no problema do turno dos enfermeiros, que a preferência pelo turno com que fica o enfermeiro mais desagradado (pior preferência, maio valor) seja o menor possível.

Relativamente ao modelo de afectação, existem as seguintes variáveis de decisão adicioanais

$z - maior\ custo\ de\ uma\ afectação\ seleccionada$

$v_i - custo\ da\ afectação\ seleccionada\ para\ o\ agente\ i, i = 1, \dots, n$

Restrições adicionais

$$v_i = \sum_{j=1}^{n} c_{ij} x_{ij}\ , i = 1, \dots, n$$

$$z \geq v_i, i = 1, \dots, n$$

Função objectivo

$$Min\ z$$

Note-se que as variáveis v_i são introduzidas para clareza do modelo que poderia penas incluir $z \geq \sum_{j=1}^{n} c_{ij} x_{ij}\ , i = 1, \dots, n$.

9.2.3 Afectação generalizada

No problema de afectação generalizada, também existem m agentes para executar n tarefas, mas cada agente i tem uma capacidade b_i e cada tarefa é executada exactamente por um agente. A tarefa j ser executada pelo agente i corresponde a um custo de c_{ij} e a um consumo de w_{ij} unidades da sua capacidade. Pretende-se minimizar o custo total.

As variáveis de decisão são

$$x_{ij} = \begin{cases} 1, se\ agente\ i\ efectua\ a\ tarefa\ j \\ 0, caso\ contrário \end{cases}, i = 1, \dots, m, j = 1, \dots, n$$

O modelo é

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

$$sujeito\ a:$$

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

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

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

De seguinda é dado um exemplo.

Exemplo 9.2 Produção

Na produção de um determinado artigo é necessário executar três tarefas (1,2,3). Para tal existem três máquinas, A, B e C que estão disponíveis 40, 50 e 30 minutos, respectivamente. De acordo com a máquina utilizada, a qualidade de execução de cada tarefa pode ser classificada numa escala de 1 a 5 (em que 1 é melhor do que 5). Dadas as diferentes características das máquinas e das tarefas a executar, o tempo de realização de

cada tarefa depende da máquina em que é efectuada. Os dados relativos a estes aspectos são dados tabela abaixo. Cada tarefa tem de ser executada numa única máquina. Pretende-se maximizar a qualidade total.

		Qualidade			Duração		
Tarefa		1	2	3	1	2	3
Máquina	A	3	2	1	25	30	10
	B	5	1	2	31	21	20
	C	4	5	2	32	18	17

As variáveis de decisão são

$$x_{ij} = \begin{cases} 1, se\ a\ tarefa\ j\ é\ realizada\ na\ máquina\ i \\ 0, caso\ contrário \end{cases}, i = 12,3; j = 1,2,3$$

O modelo é

$$Min\ z\ =\ 3x_{11} + 2x_{12} + x_{13} + \cdots + 5x_{32} + 2x_{33}$$
$$sujeito\ a:$$
$$25x_{11} + 30x_{12} + 10x_{13} \leq 40$$
$$31x_{21} + 21x_{22} + 20x_{23} \leq 50$$
$$32x_{31} + 18x_{22} + 17x_{33} \leq 30$$
$$x_{11} + x_{21} + x_{31} = 1$$
$$x_{12} + x_{22} + x_{32} = 1$$
$$x_{13} + x_{23} + x_{33} = 1$$
$$x_{ij} \in \{0,1\}, i = 1,2,3; j = 1,2,3$$

■

9.3 Emparelhamento de cardinalidade máxima

O problema do emparelhamento de cardinalidade máxima num grafo bipartido consiste em escolher o maior número de arestas de um grafo bipartido de tal forma que cada vértice nelas incidentes não indica em mais de uma aresta. É dado um exemplo na Figura 9.2.

Este problema é um caso particular do problema de afectação em que todas as arestas têm custo unitário. É também um caso particular do problema de emparelhamento (matching) sem pesos num grafo geral.

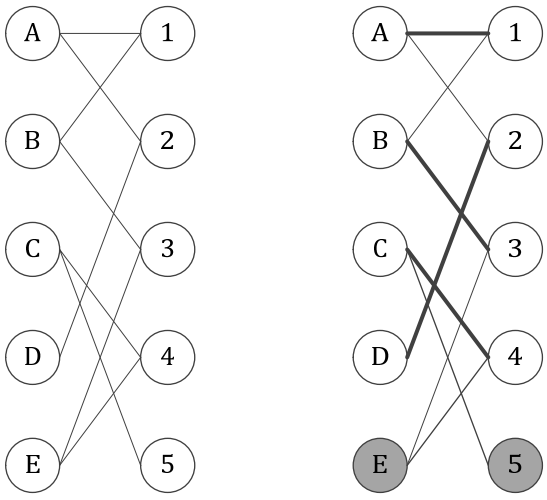


Figura 9.2. Instância de um problema de emparelhamento de cardinalidade máxima.
a) Grafo que define a instância.
b) Uma solução de valor 4.

Introduzem-se agora algumas definições que serão posteriormente utilizados algoritmicamente. Representa-se um emparelhamento por M. No exemplo, $M = \{A1, B3, C4, D2\}$. Um vértice exposto (relativo a um dado emparelhamento M) é um vértice que não faz parte do emparelhamento. Na Figura 9.2b), os vértices E e 5 são vértices expostos. Um caminho alternado (relativo a um dado emparelhamento M) é um caminho que alterna entre arestas de M e arestas que não fazem parte de M. Um caminho aumentável é um caminho alternado que tem início e final em vértices expostos.

Um caminho aumentável permite a obtenção de um emparelhamento de maior cardinalidade incertendo a pertença à solução das arestas do emparelhamento. Isto é, removendo do emparelhamento as arestas do caminho que faziam parte do emparelhamento e adicionando ao emparelhamento as arestas do caminho que não faziam parte do emparelhamento.

Um resultado fundamental é que um emparelhamento M tem cardinalidade máxima (tem o maior número de arestas possível) se e só se não existirem caminhos aumentáveis relativos a M.

No exemplo, o caminho E-4-C-5 é aumentável, obtendo-se o emparelhamento de maior cardinalidade da Figura 9.3.

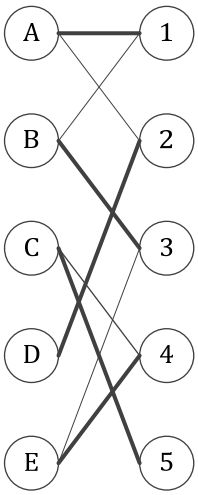


Figura 9.3. Emparelhamento resultante de um caminho aumentável.

Assim, um algoritmo para o problema do emparelhamento de cardinalidade máxima, consiste em, em cada iteração analisar todos os caminhos que partem de vértices expostos e, se algum for aumentável, obter um novo emparelhamento. Caso contrário, o algoritmo termina.

9.4 Algoritmo húngaro

O algoritmo húngaro para problemas de afectação baseia-se na transformação da matriz de custos em matrizes equivalentes (se uma afectação é óptima para a matriz original, também o é para uma matriz resultante da transformação).

Considere-se a instância da Figura 9.4.

	1	2	3	4	5
A	3	1	3	5	4
B	4	5	2	4	5
C	4	2	5	2	2
D	1	3	4	1	1
E	5	4	1	3	3

Figura 9.4. Quadro que define uma instância do problema de afectação.

A função objectivo é $Min\ z = 3x_{A1} + x_{A2} + 3x_{A3} + 5x_{A4} + 4x_{A5} + \cdots$. Dado que $x_{A1} + x_{A2} + x_{A3} + x_{A4} + x_{A5} = 1$, a função objective pode ser reescrita como $\text{Min z} = 1 + 2x_{A1} + 2x_{A3} + 4x_{A4} + 3x_{A5} + \cdots$.

Tal é equivalente a subtrair o mínimo de cada linha a todos os elementos dessa linha, obtendo-se

	1	2	3	4	5
A	2	0	2	4	3
B	2	3	0	2	3
C	0	0	3	0	0
D	0	2	3	0	0
E	4	3	0	2	2

Figura 9.5. Quadro de afectação após simplificação.

Esta transformação pode ser vista como retirando a constante 5 da função objectivo (que passa a ser $7 + 2x_{A1} + 2x_{A3} + 4x_{A4} + 3x_{A5} + 2x_{B1} + \cdots + 2x_{E4} + 2x_{E5}$).

Se existir um emparelhamento de cardinalidade 5 no grafo definido pelas arestas de custo 0, então a esse emparelhamento está associada uma afectação óptima.

O algoritmo dos caminhos aumentáveis para emparelhamento de cardinalidade máxima produz emparelhamento A2, B3, C1 e D4 (Figura 9.6). Como cardinalidade deste emparelhamento é inferior a 5, não existe nenhuma afectação óptima associada à matriz actual.

	1	2	3	4	5
A	2	0	2	4	3
B	2	3	0	2	3
C	0	0	3	0	0
D	0	2	3	0	0
E	4	3	0	2	2

Figura 9.6. Emparelhamento máximo (cinzento escuro) com arestas a cinzento.

Para prosseguir com o algoritmo é necessária uma nova transformação que é baseado em usar *c* traços (em que *c* é a cardinalidade do emparelhamento obtido anteriormente) para cobrir linhas e colunas de tal forma que todos os zeros fiquem cobertos (pelo menos por um traço). O número de traços tem de ser o menor possível.

Cada traço cobre uma linha ou uma coluna, tendo que ser usados necessariamente *c* traços como no exemplo da Figura 9.7.

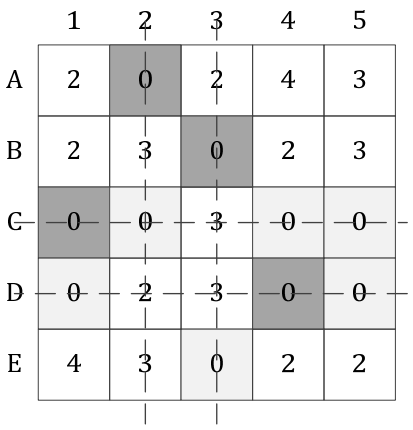


Figura 9.7. Cobertura de todas as linhas e colunas com zeros seleccionados.

Definindo I como o conjunto de linhas não cobertas e J como o conjunto de colunas não cobertas, o procedimento é o seguinte

- $\delta = min\{\bar{c}_{ij}: i \in I, j \in J\}$ onde $\bar{c}_{ij}$ é o elemento da linha i coluna j da matriz;
- Para cada célula ij tal que i $\notin I$, $j \notin J$, $\bar{c}_{ij} = \bar{c}_{ij} - \delta$ (células não cobertas);
- Para cada célula ij tal que i $\in I$, $j \in J$, $\bar{c}_{ij} = \bar{c}_{ij} + \delta$ (células cobertas duas vezes);
- Para cada célula ij tal que i $\in I$, $j \notin J$, ou i $\notin I$, $j \in J$, $\bar{c}_{ij} = \bar{c}_{ij}$ (células cobertas uma vez).

No exemplo, $\delta = 2$ e o quadro é o da Figura 9.8. O algoritmo dos caminhos aumentáveis para emparelhamento de cardinalidade máxima produz emparelhamentos A1, B3, C4 e D5. Existe o caminho aumentável E3-3B-B4-4C-C5-5D-D1-1A-A2. Como a cardinalidade do emparelhamento é 5 (a dimensão do problema), a afectação A2, B4, C5, D1, E3 é óptima com custo com 9 (Figura 9.9).

	1	2	3	4	5
A	0	0	2	2	1
B	0	3	0	0	1
C	0	2	5	0	0
D	0	4	5	0	0
E	2	3	0	0	0

Figura 9.8. Novo quadro de afectação.

	1	2	3	4	5
A	0	0	2	2	1
B	0	3	0	0	1
C	0	2	5	0	0
D	0	4	5	0	0
E	2	3	0	0	0

Figura 9.9. Afectação óptima.

Exemplo 9.3 Algoritmo húngaro

A instância é

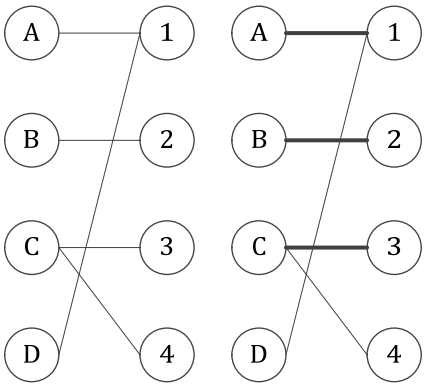
	1	2	3	4
A	*2*	*10*	*9*	*7*
B	*15*	*4*	*14*	*8*
C	*13*	*14*	*16*	*11*
D	*4*	*15*	*13*	*9*

Subtraindo o menor elemento de cada linha a todos os elementos dessa linha e Subtraindo o menor elemento de cada coluna a todos os elementos dessa coluna, obtém-se

	1	2	3	4
A	*0*	*8*	*7*	*5*
B	*11*	*0*	*10*	*4*
C	*2*	*3*	*5*	*0*
D	*0*	*11*	*9*	*5*

	1	2	3	4
A	*0*	*8*	*2*	*5*
B	*11*	*0*	*5*	*4*
C	*2*	*3*	*0*	*0*
D	*0*	*11*	*4*	*5*

O emparelhamento de cardinalidade máxima (valor 3) é



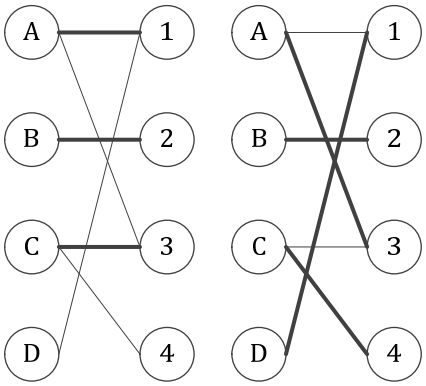
O menor número de traços para cobrir os zeros do emparelhamento é 3.

	1	2	3	4
A	0	8	2	5
B	11	0	5	4
C	2	3	0	0
D	0	11	4	5

A actualização dos coeficientes resulta em

	1	2	3	4
A	0	6	0	3
B	13	0	5	4
C	4	3	0	0
D	0	9	2	3

O emparelhamento de cardinalidade máxima é



A cardinalidade do emparelhamento de cardinalidade máxima é 4, portanto a afectação óptima é A-3, B-2, C-4, D-1 com custo total 9+4+11+4=28.

■

9.5 Exercícios

Exercício 9.1 Navios de carga

Quatro navios de carga estão disponíveis para serem usados no transporte de mercadorias de um para quatro portos. Qualquer navio pode viajar para qualquer porto mas, devido às suas características e às das cargas, o custo da viagem varia consoante o navio e o porto. Esses custos são dados na tabela seguinte (em centenas de euros).

		Porto			
		A	B	C	D
Navio	1	2	3	4	2
	2	7	2	2	6
	3	3	6	2	7
	4	6	3	6	5

Utilizando um algoritmo adequado, obtenha uma solução óptima para este problema. Indique claramente qual o navio que deve ir para cada porto e qual o valor do custo total.

Exercício 9.2 Operadores e máquinas

Numa determinada fábrica é necessário alocar um e só um operador a cada uma de seis máquinas. Para proceder a essa alocação, foi medida a produtividade (em número de peças por hora) de cada um de seis operadores em cada uma das seis máquinas. Esses valores são dados na Tabela seguinte.

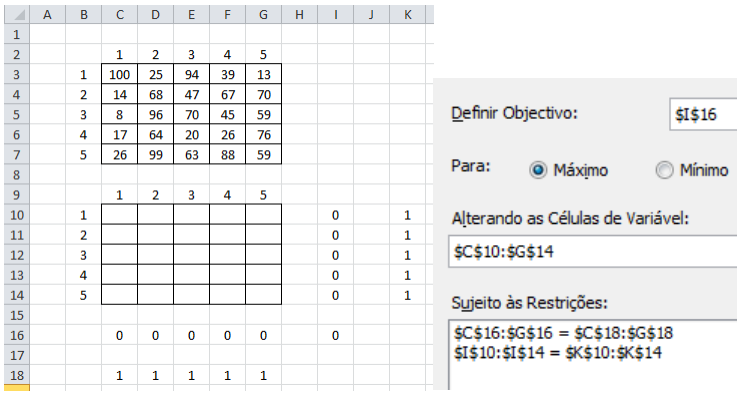
		Operador					
		A	B	C	D	E	F
Máquina	1	20	40	51	21	51	33
	2	65	95	21	88	63	68
	3	66	44	34	75	10	75
	4	31	97	90	48	98	27
	5	64	21	93	96	13	33
	6	89	5	24	9	41	63

Dado que as máquinas funcionam em paralelo, a produtividade total é a soma das produtividades em cada máquina. Determine de que forma devem ser alocados os operadores de forma a que a produtividade seja máxima.

Exercício 9.3 Projecto de investigação

Para a realização de um projecto de investigação é necessário atribuir a responsabilidade de um conjunto de 5 actividades a 5 pessoas. Cada actividade tem de ter um responsável e cada pessoa tem de ser responsável por uma actividade. Para melhor realizar essa afectação, foi estimado o grau de adequação do perfil de cada pessoa a cada actividade. Estes valores são dados nas células C3 a G7 da figura abaixo (um valor maior significa uma maior adequação). Para decidir qual a melhor afectação foi definido um modelo de programação linear que foi depois inserido no Solver do Excel conforme se pode ver nas figuras.

Para todas as células que são referidas na caixa de diálogo do solver, indique as que têm uma fórmula e qual a fórmula.



Exercício 9.4 Algoritmo húngaro

Obtenha a solução óptima do problema de afectação representado no quadro seguinte.

	1	2	3	4
A	8	11	4	3
B	9	3	9	8
C	8	7	3	5
D	8	3	9	2

Exercício 9.5 Armazém

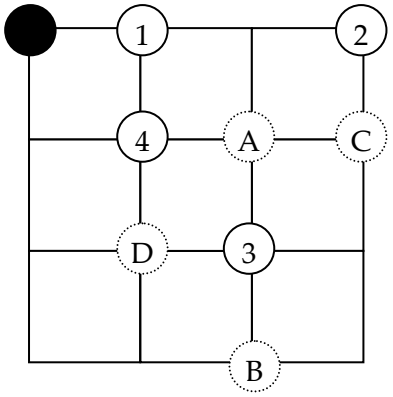
Num determinado armazém, pretendem-se recolher quatro objectos e guardar outros quatro objectos. No esquema seguinte são indicadas as posições dos quatros objectos que se pretende recolher (*1*, *2*, *3* e *4*) e as posições onde se pretende guardar os outros quatro objectos (*A*, *B*, *C* e *D*).

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*.

Como devem ser guardados e recolhidos os objectos de forma a que o veículo percorra a menor distância possível.



Exercício 9.6 Vigilância marítima

O Departamento de Transportes do Canadá (DTC) utiliza um modelo de PI para gerir os barcos que tem disponíveis para vigilância da costa do Oceano Pacífico. Considere a seguinte versão (simplificada) do problema com que se debate o DTC.

Existem três barcos (aqui designados por A, B e C) para servir as seis regiões em que a costa está dividida. Para o ano em causa, cada um dos barcos está disponível 50 semanas.

A cada região tem de ser alocado um barco. Na tabela seguinte é dado o tempo (em semanas) que cada barco serve cada região, caso a alocação em causa seja efectuada.

		Região					
		1	2	3	4	5	6
Barco	A	30	50	10	11	13	9
	B	10	20	50	10	10	17
	C	50	10	10	15	8	12

Dadas as características técnicas de cada barco e a sua localização base, os custos associados a cada barco servir cada região são muito diferentes. Na tabela seguinte são dados esses valores (em milhares de dólares canadianos).

		Região					
		1	2	3	4	5	6
Barco	A	130	30	510	30	340	20
	B	450	150	20	40	30	450
	C	40	370	120	390	40	30

A título exemplificativo, uma solução admissível para este problema é o barco A servir as regiões 1, 3 e 6 (estando ocupado 49 semanas com um custo de 660 milhares de dólares canadianos), o barco B servir as regiões 2 e 4 (estando ocupado 30 semanas com um custo de 190 milhares de dólares canadianos) e o barco C a região 1 (estando ocupado 50 semanas com um custo de 40 milhares de dólares canadianos)

Apresente um modelo de PI para o problema em causa, tendo como objectivo a minimização do custo total.

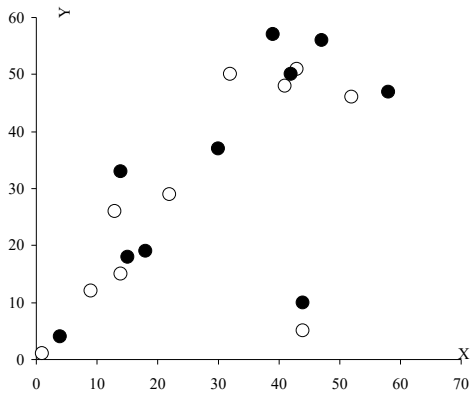
Exercício 9.7 Veículos

Uma empresa de transportes pretende decidir quais os veículos da sua frota que necessita para fazer entregas a sete clientes. Duas encomendas têm 7 m^3, três têm 3 m^3 e duas têm 2 m^3. A empresa tem quatro veículos, dois com capacidade de 9 m^3 e dois com capacidade de 8 m^3.

Com base num modelo de programação inteira, indique o menor número de veículos que permite a entrega de todas as encomendas.

Exercício 9.8 Objectos em movimento

Em vários contextos, pode ser desejável estimar a velocidade e/ou a direcção de um grupo de objectos em movimento (por exemplo, aviões, mísseis, manchas de crude no mar) com base em duas fotografias tiradas em dois momentos consecutivos. Considere o exemplo com 10 objectos representado na figura. A posição dos 10 objectos no primeiro momento é representada por um círculo branco e posição dos 10 objectos no segundo momento é representada por um círculo negro.



Uma forma de caracterizar o movimento do conjunto de objectos, é estabelecer uma associação entre cada círculo branco e cada círculo negro, de forma a que a distância global seja mínima. Sabendo que as duas fotografias foram tiradas com um segundo de intervalo e que as coordenadas dos objectos, dadas na tabela, são expressas em metros, estime a velocidade média do conjunto de objectos.

Primeiro momento	
X	Y
1	1
9	12
13	26
14	15
22	29
32	50
41	48
43	51
44	5
52	46

Segundo momento	
X	Y
4	4
14	33
15	18
18	19
30	37
39	57
42	50
44	10
47	56
58	47

Exercício 9.9 Transportes de encomendas

Uma empresa pretende entregar encomendas entre os seus *5* armazéns e os seus *5* clientes. Os transportes entre os armazéns e os clientes podem ser feitos simultaneamente mas cada armazém apenas pode fazer uma entrega (a um cliente). Os tempos de transporte entre cada armazém e cada cliente são dados na tabela (em minutos).

Pretende-se concluir todas as entregas o mais rapidamente possível.

	1	2	3	4	5
A	265	260	308	335	403
B	255	243	289	382	365
C	324	404	457	459	442
D	456	434	422	475	388
E	315	306	357	434	299

Exercício 9.10 Células de fabrico

Numa fábrica, para se montar uma determinada peça final são necessários seis componentes. Cada componente é fabricado numa e numa só célula de fabrico e cada célula de fabrico apenas pode fabricar um componente. A fábrica tem seis células de

fabrico. Na tabela é indicado o tempo que cada componente demora a ser fabricado em cada célula de fabrico (em minutos).

Em que célula deve ser fabricado cada componente de forma a que o tempo de produção de uma peça final seja o menor possível?

Célula de fabrico	Componente A	B	C	D	E	F
1	20	40	51	21	51	33
2	65	95	21	88	63	68
3	66	44	34	75	10	75
4	31	97	90	48	98	27
5	64	21	93	96	13	33
6	89	5	24	9	41	63

Exercício 9.11 Armazenamento

Pretendem-se guardar 100 objectos num determinado armazém automático com 20 locais de armazenamento. Cada um dos objectos tem um volume de c_i metros cúbicos *(i=1,...,100)*. Cada um dos locais de armazenamento tem uma capacidade de b_j metros cúbicos (*j=1,...,20*) e fica a uma distância de d_j metros *(j=1,...,20)* do local onde inicialmente se encontram os objectos. O veículo que efectua o transporte do local onde inicialmente se encontram os objectos para um dos locais de armazenamento apenas pode transportar um objecto de cada vez.

Apresente um modelo de PI que permita a minimização da distância total percorrida com o armazenamento de todos os objectos.

9.6 Bibliografia

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. Naval Research Logistics (NRL), 2(1-2), 83-97.

Pentico, D. W. (2007). Assignment problems: A golden anniversary survey. European Journal of Operational Research, 176(2), 774-793.

9.7 Resultados de aprendizagem

- Identificar e modelar através de programação inteira problemas de afectação.
- Aplicar o algoritmo húngaro para resolver instâncias de pequena dimensão de problemas de afectação.