Modelos de Investigação Operacional

J.M. Valério de Carvalho

13 de Novembro de 2020

Capítulo 1

Introdução

Somos muitas vezes confrontados com situações em que pretendemos, por exemplo, identificar como se pode atingir um determinado objectivo ou nível de serviço da forma mais eficiente, ou então como se devem usar os recursos de que dispomos para deles tirar o melhor proveito. A Investigação Operacional (IO) é uma disciplina que usa métodos analíticos para ajudar a encontrar as melhores decisões. Um slogan que captura muito bem este propósito é o usado pela associação americana de IO (INFORMS): *"Operations Research: The Science of Better®"*[1].

A Investigação Operacional também é conhecida por Ciências de Gestão *(Management Sciences)* ou Ciências de Decisão *(Decision Sciences)*. Há associações de profissionais e investigadores desta área no mundo inteiro, e também federações de associações em vários continentes. A sigla INFORMS significa *Institute for Operations Research and the Management Sciences*[2]. A APDIO (Associação Portuguesa de Investigação Operacional) [3] é um membro do *EURO - The Association of European Operational Research* [4].

Ferramentas computacionais de IO são hoje em dia utilizadas rotineiramente no mundo real, em áreas como a logística e distribuição, telecomunicações e redes de comunicação, gestão da cadeia de abastecimento, gestão de serviços de saúde, planeamento da operação de companhias de transporte (aéreo, caminho de ferro, urbano), planeamento da produção, gestão de projectos, corte e empacotamento, gestão de pessoal, gestão de florestas e muitas outras, capacitando os gestores para decisões que normalmente trazem benefícios muito significativos.

Um exemplo do uso da IO é o planeamento de operações de companhias áreas. Cada companhia opera um conjunto de rotas, que se desdobram em vôos em horas e dias da semana bem definidos. Os recursos de que as companhias dispõem são escassos, e sujeitos a condicionamentos. Os aviões são de vários tipos e de diferentes capacidades. As equipas de pilotos e as de tripulação têm jornadas laborais com durações máximas e períodos mínimos de descanso obrigatório, que podem levar a que tenham de pernoitar longe da sua base. O número e a duração dessas pernoitas estão também sujeitos a restrições semanais e mensais, podendo ser útil, ou até necessário, transferir uma equipa para um outro aeroporto, num vôo da companhia e sem prestar serviço, para aí iniciar novamente funções. Um problema relevante para a operação é estabelecer quais as equipas de pilotos e de tripulação que operam cada vôo. As companhias procuram um plano que minimize os custos de operação, e só são admissíveis os que respeitem todos os condicionamentos.

[1] http://www.scienceofbetter.org/
[2] http://www.informs.org/
[3] http://apdio.pt/home
[4] http://www.euro-online.org

Trata-se um problema complexo, mas há modelos matemáticos e técnicas de IO (algumas de natureza avançada) que permitem encontrar planos de operação de muito boa qualidade. Apesar de poderem não ser óptimos, dada a dificuldade inerente ao problema, tipicamente têm um custo que difere uma pequena percentagem (*e.g.*, 1% ou 2%) do custo óptimo do modelo matemático, e isso pode ser comprovado com informação dada pelo próprio modelo. Quando começaram a utilizá-los, as companhias áreas conseguiram reduzir substancialmente os seus custos de operação, o que depois se traduziu numa redução real dos preços dos bilhetes de avião. Um exemplo da aplicação da IO para efectuar o emparelhamento de tripulações na Air France é apresentado em [3].

Estrutura dos modelos de IO

Em geral, um modelo é uma representação de um sistema real que permite analisar e avaliar o seu comportamento. No contexto da IO, em que o propósito é identificar as melhores decisões, o sistema é representado pela descrição do espaço de decisões admissíveis. Em complemento, usa-se uma medida de eficiência que serve para associar um valor a cada decisão admissível. Para identificar, de entre todas as decisões admissíveis, qual a melhor, ou as melhores decisões, usam-se técnicas de IO que pesquisam o espaço de decisões admissíveis, um processo que se designa por "resolver o modelo". As melhores decisões encontradas são depois traduzidas num conjunto de acções no mundo real. Todos estes aspectos são discutidos de seguida.

Em sistemas reais, há latitude de decisão, mas há sempre circunstâncias que estabelecem limites às decisões, decorrentes, por exemplo, de os recursos disponíveis serem finitos, de haver regras a que temos de obedecer ou de outras limitações de vários tipos. Por isso, o espaço de decisões admissíveis (também designado por espaço de soluções) é tipicamente limitado. A definição do espaço de soluções é a primeira componente do modelo.

Em complemento, a segunda componente dos modelos matemáticos é a medida de eficiência que serve para associar um valor a cada decisão admissível. Nos modelos de IO, isso é feito através de uma função designada por função objectivo, que associa a qualquer decisão (solução) admissível (que é o argumento da função) um valor da função, que é único. Uma função objectivo normalmente representa um lucro ou um custo (associado a uma decisão).

A Programação Linear (PL) é uma área que está na base de muitas outras técnicas de IO. Foi uma das que estiveram na sua génese, e é uma das que mais contribuem para a sua visibilidade. O sentido da designação Programação Linear é o seguinte: "programação"é usado no sentido de planeamento, como em programação de actividades, e "linear", porque todas as funções envolvidas nos modelos devem ser funções polinomiais de primeiro grau (também designadas por funções afins), não sendo permitido usar funções que envolvam o produto de variáveis, como, por exemplo, as funções $3xy$ ou $5z^2$. As ferramentas de optimização que vamos usar não resolvem problemas com funções não-lineares. Doravante usaremos também "função linear"para designar uma função afim.

Construção e resolução de modelos de IO

A construção do modelo inicia-se com várias etapas preliminares. Em primeiro lugar, a definição do problema, que envolve analisar o sistema, delimitar a área de actuação, *i.e.*, identificar o que podemos decidir e o que não podemos, e definir objectivos. Em segundo lugar, a observação do sistema e a recolha de dados. A observação do sistema permite conhecer as regras do sistema e como elas condicionam as decisões e estabelecer a forma de associar um valor a cada decisão. A

recolha de dados é feita depois de identificar os que são necessários, para se ter toda a informação. Quando se faz um análise de sensibilidade, após determinar a solução óptima, os dados do problema são tratados como parâmetros, e por isso também é habitual designá-los assim.

O passo chave na construção de um modelo é a formulação. Formular um modelo de Programação Linear envolve encontrar uma forma de representar as decisões admissíveis através de variáveis de decisão (*e.g.*, reais, inteiras, binárias). São exemplos de variáveis de decisão a quantidade a produzir de um artigo, a rota a percorrer por um veículo (que pode ser percorrida ou não), o fluxo a enviar pelo arco duma rede, a actividade (ou projecto) a seleccionar e o instante de tempo em que se deve iniciar a execução de uma actividade.

Mas a formulação só é válida se se conseguir: i) exprimir as regras gerais de funcionamento e tudo o que condiciona as decisões admissíveis através de restrições que envolvem funções lineares dessas variáveis de decisão; e ii) exprimir a forma de atribuir um valor a cada decisão admissível através de uma função objectivo, que também deve ser uma função linear das variáveis de decisão. Se tal não for possível, deve procurar-se uma formulação com outras variáveis de decisão.

Uma restrição é uma afirmação (ou declaração) matemática, *viz.* uma equação ou uma inequação (também designada por desigualdade), envolvendo uma função das variáveis de decisão. São exemplos a declaração da relação existente entre a função que exprime a quantidade de um recurso que as actividades consomem e o valor do recurso disponível (que é um parâmetro do problema); ou a declaração da relação existente entre a função que exprime a quantidade de um bem que as actividades produzem e número de unidades desse bem pedidas por um cliente (um parâmetro do problema); ou da relação que traduz as regras de funcionamento do sistema (e.g., conservação de fluxo, precedência entre operações).

A função objectivo é uma função linear que pretendemos maximizar ou minimizar, e é única. Há uma área de IO em que se estudam problemas tendo em consideração mais do que um objectivo, designada por programação multi-objectivo, que está fora do âmbito desta UC.

A resolução de um modelo visa encontrar a melhor decisão. Para encontrar a solução óptima, são usadas ferramentas de optimização (*e.g.*, *LPSolve, Relax4, IBM ILOG Cplex, Gurobi*), que aceitam ficheiros de *input* com formatos definidos. Estas ferramentas usam métodos de IO, para resolver problemas de programação linear, fluxos em rede e programação linear inteira. Sugere-se a utilização do *LPsolve* e *Relax4*, que são *freeware* e disponibilizados no Blackboard.

Alguns aspectos a ter em conta

Há quem diga que construir modelos é uma arte. Se isso é verdade, também se pode beneficiar do conhecimento do quadro de trabalho e da análise de exemplos. O objectivo deste documento é não apenas apresentar formulações e modelos, mas também mostrar a aplicação da metodologia de construção dos modelos.

Em geral, eles são ilustrados com exemplos de aplicação, e, nalgumas situações, apresentam-se também o ficheiro correspondente de *input* a submeter aos *packages* de resolução, *LPsolve* e *Relax4*, e os ficheiros de *output* desses *packages*, que indicam a solução óptima do modelo, *i.e.*, o valor óptimo da função objectivo e os valores óptimos das variáveis de decisão. Adicionalmente, o resultado é interpretado, traduzindo os valores das variáveis de decisão em acções no sistema real.

Um aspecto importante de que nos devemos aperceber é que um problema pode ter formulações diferentes. Diferentes escolhas de variáveis de decisão conduzem a modelos diferentes. Para o ilustrar, apresentam-se três formulações do problema de corte e empacotamento a uma dimensão que conduzem a três modelos diferentes. Embora a solução óptima de cada um desses modelos seja expressa em termos de variáveis de decisão diferentes, o valor das diferentes soluções ópti-

mas é igual, e a tradução dessas soluções dá origem a acções no mundo real que são naturalmente equivalentes.

Há um outro aspecto que devemos ter em consideração em situações em que se analisa um sistema complexo, tendo em vista construir um modelo de IO. Trata-se de uma questão de ordem geral, a que faremos referência apenas aqui neste documento. Esse aspecto prático é o detalhe do modelo. Um modelo é uma representação da realidade, pelo que é sempre uma aproximação. É necessário balancear a qualidade de aproximação e a dimensão do modelo. Por um lado, se o modelo não for uma boa aproximação da realidade, os resultados obtidos podem não ser úteis para o sistema real. Por outro lado, se tivermos em consideração demasiados detalhes do sistema real, a dimensão de um modelo pode crescer de tal forma que se torna impraticável resolvê-lo.

Complexidade de problemas

O aspecto que acabámos referir está relacionado com um outro que não faz parte da matéria em avaliação na UC, e que é determinante na avaliação da magnitude dos modelos que podemos resolver em tempo útil. É a complexidade dos problemas. Iremos fazer referência ao facto de que há problemas mais difíceis de resolver do que outros. Enquanto que os problemas de programação linear e os problemas de fluxo em rede são fáceis, no sentido de que existem algoritmos polinomiais para a sua resolução, sendo possível obter soluções óptimas para instâncias de muito grande dimensão, os problemas de programa inteira são difíceis, no sentido de que não são conhecidos algoritmos polinomiais para a sua resolução.

Isto não quer dizer que, para problemas de programação inteira, não é possível obter soluções óptimas ou de muito boa qualidade para instâncias de dimensão razoável provenientes do problemas do mundo real em tempo razoável. De facto, isso é possível, e a prática demonstra-o. Aspectos relacionados com a dificuldade (complexidade) dos problemas são abordados com detalhe no Anexo A.

Ainda com respeito aos problemas de programação inteira, que, no pior caso, podem ser muito difíceis de resolver, a prática mostra que os seus graus de dificuldade são diferentes. Os modelos com variáveis de decisão inteiras apenas (sem variáveis binárias) são, na prática, quase tão fáceis de resolver como os modelos de programação linear, apesar da sua inerente dificuldade. Os modelos mais difíceis de resolver são os de programação inteira com restrições lógicas, que envolvem o uso de variáveis de decisão binárias, quando têm uma dimensão apreciável.

Mas, mesmo a estes, aplica-se o que foi dito. Quando não se consegue determinar a solução óptima, as técnicas de pesquisa do espaço de soluções admissíveis, que são métodos de enumeração inteligente, permitem tipicamente encontrar, em tempo razoável, soluções de muito boa qualidade, com valores que diferem apenas uma pequena percentagem do valor da solução óptima.

Organização do documento

Este documento está organizado da seguinte forma. No segundo capítulo, apresentam-se modelos de programação linear e inteira. Os modelos de programação inteira usam variáveis inteiras ou binárias com relações simples. No Capítulo 3, apresenta-se o modelo geral de problemas de fluxo em rede, e alguns problemas típicos de optimização combinatória que podem ser formulados como problemas de programação linear. Finalmente, no Capítulo 4, apresentam-se modelos de programação inteira com restrições lógicas, em que são utilizadas variáveis binárias para modelar situações complexas, como, por exemplo, aproximações lineares de sistemas reais não-lineares. Finalmente, no Anexo A, aborda-se o tema da complexidade de problemas, matéria que não faz

parte da matéria em avaliação nesta UC. Além disso, e apenas a título informativo, é indicada a complexidade de cada problema cujo modelo é apresentado.

Capítulo 2

Programação linear

2.1 Modelo geral

O problema de optimização (ou maximização ou minimização) de uma função objectivo linear sujeita a restrições lineares pode ser expresso na forma geral:

$$\begin{aligned} \{\max, \min\}\ z \quad &= \quad \sum_{j=1}^{n} c_j x_j \\ \text{suj. a} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \forall\, i \in R_{\leq} \\ & \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \forall\, i \in R_{\geq} \\ & \sum_{j=1}^{n} a_{ij} x_j = b_i, \forall\, i \in R_{=} \\ & x_j \geq 0, j = 1, \ldots, n, \end{aligned}$$

sendo:

- z : função objectivo
- $x_1, \ldots, x_j, \ldots, x_n$: variáveis de decisão.
- $c_1, \ldots, c_j, \ldots, c_n$: coeficientes da função objectivo $\sum_{j=1}^{n} c_j x_j$.
- b_i : coeficiente do lado direito das restrições, indexado por i
- $R_{\leq}, R_{\geq}, R_{=}$: conjuntos de restrições do tipo $\leq, \geq, =$, respectivamente.
- $x_j \geq 0$: restrições de não-negatividade, para todos os j.

Notar que os valores das variáveis de decisão são sempre não-negativos. Em situações em que se pretendam usar variáveis que possam também assumir valores negativos, *i.e.*, variáveis sem restrição de sinal, podemos sempre usar a transformação básica indicada na Secção 2.11.

Assumpções

Um modelo é uma representação da realidade. A validade de um modelo de PL (que usa funções lineares na função objectivo e nas restrições) resulta da assumpção de que são válidos os seguintes pressupostos:

- Proporcionalidade: os contributos para a função objectivo e para as funções usadas nas restrições são proporcionais aos valores das variáveis de decisão.
- Aditividade: em cada função, o contributo total é dado pela soma dos contributos das diferentes variáveis.
- Divisibilidade: as variáveis de decisão podem tomar qualquer valor, incluindo valores fraccionários.
- Determinismo: os dados são valores conhecidos e constantes.

O uso de variáveis binárias no Capítulo 4 - Programação inteira com restrições lógicas - ajuda a modelar sistemas em que a assumpção de um (ou mais) destes pressupostos viola de uma forma flagrante o que ocorre na realidade.

Apresentação de modelos

A apresentação de um modelo de IO deve incluir a formulação e o modelo propriamente dito, e a comprovação de que o modelo é uma representação adequada do sistema real. A formulação do problema deve conter, usando linguagem corrente e em linhas gerais:

1. a descrição do problema, indicando aspectos relevantes, como os recursos disponíveis ou regras gerais de funcionamento;
2. o objectivo;
3. a escolha das variáveis de decisão e o modo como os valores das variáveis de decisão (ou conjuntos de valores) se traduzem em decisões a implementar no sistema real;
4. uma apresentação da coerência global do modelo a construir, *i.e.*, como se pretende representar o problema do sistema real através de um modelo com restrições lineares e uma função objectivo linear;
5. em casos mais complexos em que a formulação não é de todo evidente, por ser justificada por resultados teóricos, estes devem ser apresentados, ou, em alternativa, deve ser fornecida evidência suficiente, complementada por referências bibliográficas.

A apresentação do modelo é feita usando linguagem matemática e inclui a descrição de cada elemento do modelo (eventualmente agrupando elementos) deverá ser justificada e acompanhada de uma explicação com detalhe do seu significado, e de uma identificação da sua dimensão.

1. variáveis de decisão (*e.g.*, x_i : número de artigos de tipo i a produzir por hora, $i = 1,2$). Adicionalmente, deverá haver uma declaração do tipo de variável, por exemplo:

   - $x_1, x_2 \geq 0$ (ou $x_1, x_2 \in \mathbb{R}_{\geq 0}$ ou $x \in \mathbb{R}^2_{\geq 0}$);

- $x_1, x_2 \geq 0$ e inteiros (ou $x_1, x_2 \in \mathbb{Z}_{\geq 0}$ ou $x \in \mathbb{Z}^2_{\geq 0}$); ou
- $x_1, x_2 \in \{0,1\}$ (ou $x_1, x_2 \in \mathbb{B}$ ou $x \in \mathbb{B}^2$ ou $x \in \{0,1\}^2$).

2. parâmetros (*e.g.*, número de artigos disponíveis/mês, número de horas da máquina j disponíveis / dia, $j = 1, \ldots, m$, etc.);
3. função objectivo (linear) (*e.g.*, lucro diário), explicitando se se trata de um problema de maximização ou minimização;
4. restrições (lineares), eventualmente com uma verificação da coerência dimensional das funções lineares das restrições quando elas se relacionam com dados do problema (*e.g.*, horas da máquina j / dia, $j = 1, \ldots, m$, etc.).

A apresentação do modelo deve ser complementada, mostrando como é que:

1. cada restrição (ou cada grupo de restrições) traduz as regras de funcionamento do sistema;
2. o conjunto das restrições formam um conjunto coerente, definindo desse modo um espaço de soluções desejado, correspondendo ao espaço de decisões admissíveis no sistema real;
3. a função objectivo traduz a medida desejada de eficiência do sistema;

2.2 Plano de produção

O problema que vamos abordar envolve decisões a um nível táctico. O objectivo do modelo é seleccionar os artigos (e as quantidades desses artigos) que devemos produzir, de modo a tirar o maior proveito do conjunto de recursos disponíveis no sistema produtivo.

Após determinar a solução óptima ao nível táctico, é necessário definir um plano que estabeleça quando devem ser executadas as operações de produção dos artigos, ao longo do tempo. Esse plano é designado de plano operacional. Determinar o melhor plano operacional é um problema diferente. O seu objectivo é definir a sequência de execução das operações em cada máquina, a atribuição de trabalhadores a máquinas, e todos os detalhes envolvidos nas operações, como, por exemplo, operações de preparação de máquinas para a produção de lotes de artigo do mesmo tipo.

Há também modelos para estes problemas operacionais. No entanto, são modelos mais complexos, porque considerar os aspectos referidos pode violar as assumpções de Programação Linear. Por isso, para garantir que é possível encontrar um plano operacional admissível para a produção dos artigos decidida ao nível táctico, é habitual entrar em linha de conta, no problema táctico, com margens de segurança (*e.g.*, tempos de produção ligeiramente mais elevados do que os efectivamente requeridos a nível operacional).

Exemplo 2.1. Uma empresa produz 2 tipos de artigos: Artigo 1 e Artigo 2. A produção destes artigos requer 3 tipos de recursos: Material, Mão de Obra e Tempo-Máquina. O objectivo é determinar o **plano de produção diário** (solução admissível) que **maximiza o lucro total** (com o valor óptimo).

A quantidade disponível de cada recurso, o consumo de recursos por cada artigo produzido e o lucro líquido de cada artigo são os seguintes:

	Artigo 1	Artigo 2	Quantidade disponível
Material	3 [unid./art.]	2 [unid./art.]	120 [unid./dia]
Tempo-Homem	1 [h.hom./art.]	2 [h.hom./art.]	80 [h.hom./dia]
Tempo-Máquina	1 [h.maq./art.]	0 [h.maq./art.]	30 [h.maq/dia]
Lucro Unitário	12 [U.M./art.]	10 [U.M./art.]	

Em primeiro lugar, devemos identificar os parâmetros (ou dados do sistema que não podem ser alterados). Eles são:

- quantidade disponível de cada recurso;
- lucro unitário dos artigos;
- consumo de recursos por cada artigo (coeficientes tecnológicos)

A escolha de variáveis de decisão deve ser tal que permita construir funções lineares das variáveis de decisão para exprimir a função objectivo (função lucro) e as funções que traduzam o uso de recursos. Neste exemplo, isso pode ser feito usando as seguintes variáveis de decisão:

- x_1 : quantidade de artigos de tipo 1 a fabricar diariamente [art./dia]
- x_2 : quantidade de artigos de tipo 2 a fabricar diariamente [art./dia]

Uma restrição é usada para definir as decisões admissíveis; é uma relação entre uma **função linear** das variáveis de decisão e uma constante. Podemos construir funções lineares que traduzem a forma como usamos os recursos. Vamos ver exemplos:

- a função linear $3x_1$ traduz a quantidade de material usada diariamente [unid./dia] para produzir artigos de tipo 1.
- a função linear $2x_2$ traduz a quantidade de material usada diariamente [unid./dia] para produzir artigos de tipo 2.
- a função linear $3x_1 + 2x_2$ traduz a quantidade de material usada diariamente [unid./dia] para produzir artigos de ambos os tipos.

Uma vez definida a função linear que traduz a quantidade de material usada para produzir artigos de ambos os tipos, estamos em condições para estabelecer a restrição:

- a restrição $3x_1 + 2x_2 \leq 120$ estabelece que apenas são admissíveis as soluções em que o uso de recursos não exceda a disponibilidade diária de material [unid./dia]

Um dos aspectos que é importante na validação de um modelo é a verificação da consistência das relações matemáticas. No exemplo apresentado, ambos os lados da inequação (desigualdade) se exprimem na mesma unidade [unid./dia].

O mesmo se passa com a função objectivo. A função objectivo associa um valor a cada solução; deve ser uma **função linear** das variáveis de decisão:

- função objectivo $12x_1 + 10x_2$: lucro diário [U.M./dia]

O modelo de programação linear é o seguinte:

- Função objectivo:

$$\max z = 12x_1 + 10x_2$$

- Restrições:

$$\begin{array}{lcl} 3x_1 + 2x_2 & \leq & 120 \\ 1x_1 + 2x_2 & \leq & 80 \\ 1x_1 & \leq & 30 \\ x_1, x_2 & \geq & 0 \end{array}$$

Neste exemplo, todas as variáveis têm restrições de não-negatividade ($x_1, x_2 \geq 0$). Dada ser esse o caso geral, não é necessário declarar as variáveis como não-negativas [1] no ficheiro de *input* do LPSolve, que é o seguinte:

```
/* função objectivo */
max:   12 x1 + 10 x2;

/* restrições */
material: 3 x1 + 2 x2 <= 120;
maodobra: 1 x1 + 2 x2 <=  80;
tmaquina: 1 x1        <=  30;
```

Resolvendo o modelo com o LPSolve, obtém-se o seguinte relatório com a solução óptima:

Objective	
Variables	result
	540
x1	20
x2	30

que equivale a fazer 20 unid./dia de produto 1 e 30 unid./dia de produto 2, com um lucro diário de 540 U.M..

2.3 Problema da dieta

Um avicultor pretende determinar a quantidade que deve utilizar de cada alimento disponível no mercado, de modo a satisfazer as necessidades nutricionais das suas aves, minimizando o custo de alimentação diário. Seja $J = \{1, \ldots, j, \ldots, n\}$ o conjunto de rações disponíveis no mercado, e seja c_j o custo unitário da ração $j, j \in J$. Este problema pode ser formulado usando variáveis de decisão $x_j, j \in J$, que representam a quantidade diária de alimento j a dar às aves.

É necessário satisfazer m requisitos nutriciais, indexados por i. O coeficiente a_{ij} indica a quantidade de nutriente i na ração j. O modelo do problema da dieta é o seguinte:

$$\min \quad \sum_{j \in J} c_j x_j \tag{2.1}$$

$$\text{suj. a} \quad \sum_{j \in J} a_{ij} x_j \geq b_i, \; i = 1, 2, \ldots, m \tag{2.2}$$

$$x_j \geq 0, \; \forall j \in J \tag{2.3}$$

Exemplo 2.2. Os nutrientes, o custo de cada alimento e as necessidades mínimas diárias são os apresentados no seguinte quadro.

[1] Para declarar variáveis sem restrição de sinal, usar, *e.g.*, `free x1, x2;` na última linha do ficheiro de *input*.

	alimentos			mínimo
nutriente	milho	trigo	ração	diário
proteínas	4	8	4	10
hidratos de carbono	2	4	4	6
vitaminas	3	2	4	4
custo (U.M.)	0.10	0.06	0.04	

Dados:

- b_i : quantidade mínima diária do nutriente i
- c_j : custo do alimento j
- a_{ij} : quantidade de nutriente i existente na unidade de peso do alimento j

Variáveis de decisão:

- x_1 : quantidade de milho diária.
- x_2 : quantidade de trigo diária.
- x_3 : quantidade de ração.

O modelo do problema da dieta para este exemplo é:

$$\begin{array}{rrrrrl} \min z = & 0.10x_1 & +0.06x_2 & +0.04x_3 & & \\ & 4x_1 & +8x_2 & +4x_3 & \geq 10 \\ & 2x_1 & +4x_2 & +4x_3 & \geq 6 \\ & 3x_1 & +2x_2 & +4x_3 & \geq 4 \\ & x_1, x_2, x_3 \geq 0 & & & \end{array}$$

O ficheiro de input do LPSolve é o seguinte:

```
////////////////////////////////////
//      Problema da dieta
////////////////////////////////////

/* Função objectivo  */
min: 0.10 x1 + 0.06 x2 +0.04 x3;

/* Restrições */
4 x1 + 8 x2 + 4 x3 >= 10;
2 x1 + 4 x2 + 4 x3 >=  6;
3 x1 + 2 x2 + 4 x3 >=  4;
```

A solução óptima deste exemplo, fornecida pelo LPSolve, significa que se deve usar diariamente uma unidade do alimento 2 e meia unidade do alimento 3, sendo o custo óptimo igual a 0.08.

Variables	result
	0,08
x1	0
x2	1
x3	0,5

□

2.4 Problema do saco de mochila: formulação I

O problema do saco de mochila consiste em maximizar o lucro (utilidade) dos itens seleccionados para incluir num saco de mochila. Considerem-se variáveis de decisão x_j, que significam o número de itens do tipo j a incluir no saco de mochila. A cada item está associado um lucro p_j, e um peso w_j. A mochila tem uma capacidade W. A formulação do problema do saco de mochila, designado na literatura anglo-saxónica por *knapsack problem,* é a seguinte:

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} p_j x_j \\ \text{suj. a} \quad & \sum_{j=1}^{n} w_j x_j \leq W \\ & x_j \geq 0 \text{ e inteiro, } j = 1,2,\ldots,n \end{aligned}$$

Exemplo 2.3. Um investidor pretende aplicar 90 U.M. em valores mobiliários. Numa primeira fase, foram seleccionadas cinco companhias. As acções de cada companhia são vendidas em lotes. O valor e o retorno esperado de cada lote são os indicados na seguinte Tabela:

companhia	1	2	3	4	5
retorno	50	80	14	18	20
valor	12	20	4	6	8

Este problema pode ser formulado como o seguinte problema de saco de mochila com variáveis inteiras gerais, que permitem seleccionar um, ou mais, lotes de cada tipo de acções:

$$\begin{aligned} \max\ & 50x_1 + 80x_2 + 14x_3 + 18x_4 + 20x_5 \\ \text{suj. a}\ & 12x_1 + 20x_2 + 4x_3 + 6x_4 + 8x_5 \leq 90 \\ & x_j \geq 0 \text{ e inteiro, } j = 1,2,\ldots,n \end{aligned}$$

O modelo do problema é o seguinte:

```
/* função objectivo */
max: 50 x1 + 80 x2 + 14 x3 + 18 x4 + 20 x5;

/* Restrição */
12 x1 + 20 x2 + 4 x3 + 6 x4 + 8 x5 <= 90;

/* Restrições de integralidade */
```

```
int x1,x2,x3,x4,x5;
```

A solução óptima deste exemplo, fornecida pelo LPSolve, significa que se devem adquirir 7 lotes da companhia 1 e um lote da companhia 4, sendo o valor esperado do retorno óptimo igual a 368. A resolução deste exemplo envolveu o uso do algoritmo de partição e avaliação, um método de enumeração implícita usado na resolução de problemas de programação inteira. No processo de enumeração, foram encontradas 5 soluções admissíveis para o problema inteiro, sucessivamente melhores, até o algoritmo terminar, reconhecendo que a última solução é a solução óptima.

Variables	MILP Feasible	MILP Better	MILP Better	MILP Better	MILP Better	result
	352	358	360	362	368	368
x1	0	2	4	5	7	7
x2	4	3	2	1	0	0
x3	1	0	0	1	0	0
x4	1	1	0	1	1	1
x5	0	0	0	0	0	0

□

O problema de saco de mochila tem apenas uma restrição. Quando há restrições relativas a vários tipos de recursos, obtém-se um problema que é designado por problema de saco de mochila multidimensional.

2.5 Problemas com colecções de subconjuntos

Existe um conjunto muito significativo de problemas de programação inteira em que é estabelecida uma relação entre um conjunto e uma colecção de subconjuntos formada a partir desse conjunto. Em muitos desses problemas, verifica-se que não existe nenhuma estrutura especial na forma dos subconjuntos que constituem a colecção, sendo necessário formular os problemas recorrendo a uma enumeração de todos os subconjuntos.

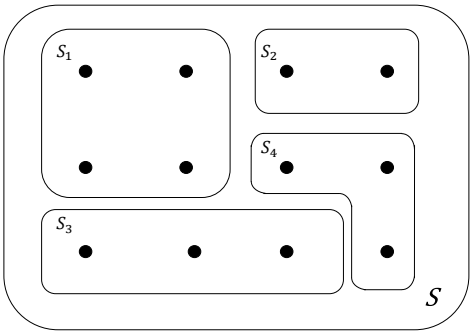
Vamos analisar 3 tipos de problemas em que as relações entre o conjunto e os subconjuntos são relações de partição, cobertura e empacotamento, respectivamente.

2.5.1 Partição de um conjunto

Seja S um conjunto finito com m elementos, e $S_1, S_2, \ldots, S_n$, uma colecção de subconjuntos de S. Uma partição do conjunto S é uma colecção desses subconjuntos, $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$, identificados pelos índices $i_1, i_2, \ldots, i_k$, tal que:

$$\begin{aligned} \cup_{j=1}^{k} S_{i_j} &= S \\ S_{i_j} \cap S_{i_k} &= \emptyset, \forall\, j,k \end{aligned}$$

Exemplo 2.4. Os subconjuntos S_1, S_2, S_3 e S_4 são uma partição dos elementos do conjunto S.



□

O problema da partição de um conjunto S, designado na literatura anglo-saxónica por *set partitioning problem,* consiste em seleccionar a partição de menor custo. Este problema pode ser formulado como um problema de programação inteira binária, com variáveis de decisão x_j, cuja coluna tem elementos $a_{ij}, i = 1,\ldots,m$, que são:

$$a_{ij} = \begin{cases} 1 & \text{, se } i \in S_j \\ 0 & \text{, caso contrário} \end{cases}$$

A formulação do problema da partição de um conjunto é a seguinte:

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j x_j \\ \text{suj. a} \quad & \sum_{j=1}^{n} a_{ij} x_j = 1,\ i = 1,2,\ldots,m \\ & x_j = 0 \text{ ou } 1,\ j = 1,2,\ldots,n \end{aligned}$$

Este problema tanto pode surgir na versão de minimização de custos ou de maximização de lucros.

Exemplo 2.5. Uma companhia pretende vender 3 lotes de terreno, identificados por A,B e C. Para o efeito, abriu um concurso em que aceitava propostas para um lote, ou para um conjunto de lotes. As 9 propostas recebidas, numeradas por ordem de chegada, são as seguintes:

Proposta	1	2	3	4	5	6	7	8	9
lote A	1	1	1	1		1		1	
lote B	1	1			1	1	1		
lote C	1		1		1				1
	12	9	7	2	7	8	4	3	4

A proposta 2, por exemplo, significa que foram oferecidas 9 U.M. pelos lotes A e B em conjunto. Trata-se de um problema de determinar a partição de maior peso, cuja formulação é apresentada na Figura 2.1. A solução óptima consiste em aceitar as propostas 2 e 9, com um valor de 13 U.M..

□

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	
lote A	1	1	1	1		1		1		= 1
lote B	1	1			1	1	1			= 1
lote C	1		1		1				1	= 1
max	12	9	7	2	7	8	4	3	4	

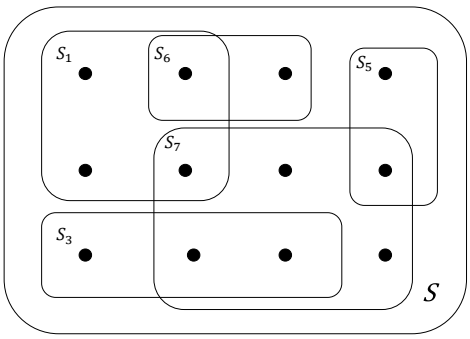
Figura 2.1: Maximização do lucro da venda de lotes

2.5.2 Cobertura de um conjunto

Outro problema importante é o problema da cobertura de um conjunto. Uma cobertura do conjunto S é uma colecção de subconjuntos que podem não ser disjuntos, *i.e.*, tal que:

$$\cup_{j=1}^{k} S_{i_j} \quad = \quad S$$

Exemplo 2.6. Os subconjuntos $S_1, S_2, \ldots, S_7$ são uma cobertura dos elementos do conjunto S.



□

O problema da cobertura de um conjunto S, designado na literatura anglo-saxónica por *set covering problem,* consiste em seleccionar a cobertura de menor custo. A sua formulação é a seguinte:

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j x_j \\ \text{suj. a} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq 1, \ i = 1,2,\ldots,m \\ & x_j = 0 \text{ ou } 1, \ j = 1,2,\ldots,n \end{aligned}$$

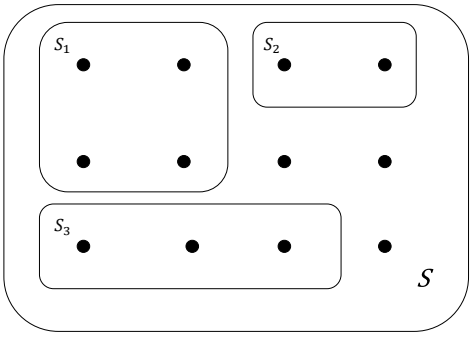
Trata-se de uma formulação semelhante à do problema de partição de um conjunto com as restrições do tipo = substituídas por restrições do tipo ≥ .

2.5.3 Empacotamento num conjunto

Finalmente, um empacotamento é uma colecção de subconjuntos disjuntos cuja reunião seja um subconjunto de S, isto é, tal que:

$$\begin{aligned} \cup_{j=1}^{k} S_{i_j} &\subseteq S \\ S_{i_j} \cap S_{i_k} &= \emptyset, \forall j,k \end{aligned}$$

Exemplo 2.7. Os subconjuntos S_1, S_2 e S_3 são um empacotamento dos elementos do conjunto S.



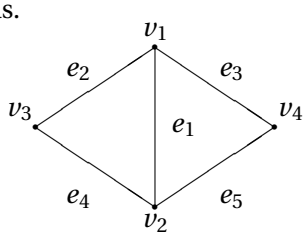
□

O problema de empacotamento, designado na literatura anglo-saxónica por *set packing problem,* consiste em seleccionar o empacotamento de maior peso. A sua formulação é a seguinte:

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ \text{suj. a} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq 1,\ i = 1,2,\ldots,m \\ & x_j = 0 \text{ ou } 1,\ j = 1,2,\ldots,n \end{aligned}$$

No caso em que há apenas dois 1's por colunas, o problema de empacotamento pode ser definido sobre um grafo, em que cada linha corresponde a um vértice e cada variável a uma aresta incidente em dois vértices. O problema de optimização correspondente é o problema de emparelhamento de peso máximo (ver secção 4.4.5).

Exemplo 2.8. A matriz $A = [a_{ij}]$ a seguir apresentada pode ser associada a um grafo com 4 vértices e com 5 arestas.



$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

□

2.6 Problemas de corte e empacotamento

Os problemas de corte e empacotamento têm uma estrutura semelhante. Na literatura anglo-saxónica, estas classes de problemas são designadas por *cutting stock problems* e *bin packing problems,* respectivamente. Iremos abordar o caso mais simples com apenas uma dimensão.

No problema de corte, pretende-se determinar o modo como um stock de matérias primas deve ser cortado em partes menores de maneira a satisfazer pedidos colocados por clientes. Dada uma quantidade ilimitada de rolos com a largura W, e dados m clientes com pedidos de b_i rolos de largura w_i, $0 < w_i \leq W, i = 1, \ldots, m$, programar os cortes a efectuar de modo a minimizar o número de rolos utilizados.

No problema de empacotamento, pretende-se determinar o modo como um conjunto de itens deve ser empacotado em contentores (*bins*) de igual capacidade, ou seja, dado um fornecimento ilimitado de *bins* de capacidade W e uma lista de n itens de dimensão w_i, $0 < w_i \leq W, i = 1, \ldots, n$, atribuir todos os itens ao menor número possível de *bins* sem exceder a capacidade de nenhum *bin*.

As semelhanças entre os dois problemas são evidentes. A Figura 2.2 mostra um exemplo de um problema de corte / empacotamento. Na perspectiva do problema de corte, nos dois últimos rolos, há desperdício de matéria prima em resultado de os itens neles colocados não ocuparem integralmente a largura do rolo, enquanto que, do ponto de vista do problema de empacotamento, há espaço não ocupado.

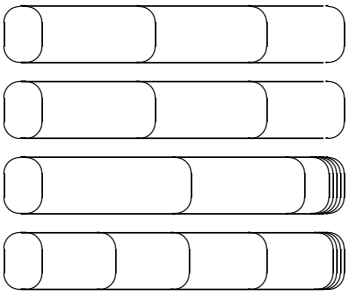


Figura 2.2: Problema de corte / empacotamento a uma dimensão

2.6.1 Formulação I - variáveis com dois índices

Esta formulação usa dois conjuntos de variáveis de decisão binárias. Cada variável binária do primeiro conjunto representa a atribuição de um item a um rolo. Essa variável define o rolo em que o item é colocado. Cada variável binária do segundo conjunto representa a utilização de um rolo de matéria prima.

A formulação do problema de empacotamento abaixo apresentada é baseada em variáveis de decisão x_{ij}, cujo significado é o seguinte:

$$x_{ij} = \begin{cases} 1 & \text{, se o item } j \text{ é atribuído ao bin } i \\ 0 & \text{, caso contrário} \end{cases}$$

Quando uma variável x_{ij} assume um valor positivo, isso significa que o bin i é utilizado para colocar o item j. As variáveis y_i contam o número de bins usados:

$$y_i = \begin{cases} 1 & \text{, se o bin } i \text{ é usado} \\ 0 & \text{, caso contrário} \end{cases}$$

A formulação do problema é a seguinte:

$$\begin{aligned} \min z &= \sum_{i=1}^{n} y_i \\ \text{suj. a} \quad & \sum_{j=1}^{n} w_j x_{ij} \leq W y_i, \ \forall i \in I \\ & \sum_{i=1}^{n} x_{ij} = 1, \ \forall j \in J \\ & y_i = 0 \text{ ou } 1, \ \forall i \\ & x_{ij} = 0 \text{ ou } 1, \ \forall i, j \end{aligned}$$

O lado esquerdo do primeiro conjunto de restrições é uma função que indica o espaço usado no rolo, que não pode exceder a largura do rolo; a função do lado direito, que é uma função linear de uma variável binária, pode tomar o valor W ou 0, consoante o rolo é utilizado ou não, respectivamente. As restrições do segundo conjunto indicam que cada item deve ser colocado exactamente num dos rolos. A função objectivo visa minimizar o número de rolos utilizados.

2.6.2 Formulação II - variáveis de padrões de corte

Esta formulação usa variáveis de decisão, cada uma delas representando uma forma diferente de combinar itens num padrão de corte, como se explica de seguida. Defina-se padrão de corte como um possível arranjo de pedidos na largura do rolo. Para o padrão de corte ser válido,

$$\begin{aligned} & \sum_{i=1}^{m} a_{ij} w_i \leq W \\ & a_{ij} \geq 0 \text{ e inteiro }, \ \forall j \in J. \end{aligned}$$

sendo a_{ij} o número de rolos de largura w_i, $0 < w_i \leq W, i = 1, \ldots, m$, obtidos a partir do padrão de corte j, e J o conjunto de padrões de corte permitidos.

Para o padrão de corte j, a perda T_j associada é:

$$T_j = W - \sum_{i=1}^{m} a_{ij} w_i.$$

Exemplo 2.9. Considere-se um conjunto de três pedidos de largura 12, 10 e 6, respectivamente. A largura dos rolos é de 30. Os padrões de corte possíveis são os apresentados na Tabela 2.2. Consideram-se apenas padrões de corte *maximais*, em que a perda é inferior à largura do menor item (ver justificação em notas).

□

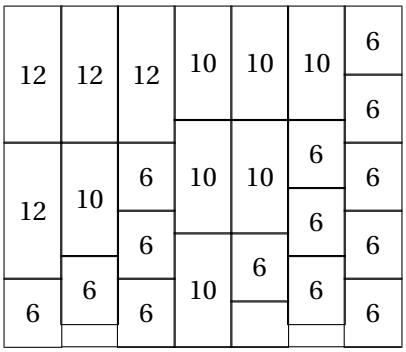


Tabela 2.1: Padrões de corte

Seja x_j uma variável de decisão que designa o número de rolos a serem cortados segundo o padrão de corte j. Defina-se uma matriz A com informação sobre os padrões de corte, *i.e.*, cada coluna $A_j = (a_{1j}, \ldots, a_{ij}, \ldots, a_{mj})^T$ define um padrão de corte, com elementos a_{ij} conforme foram definidos acima.

O problema de corte tem como objectivo seleccionar o conjunto de padrões de corte, que minimiza o número total de rolos necessários à satisfação dos pedidos de b_i rolos de largura w_i :

$$\min \quad \sum_{j \in J} x_j \tag{2.4}$$

$$\text{suj. a} \quad \sum_{j \in J} a_{ij} x_j \geq b_i, \ i = 1, 2, \ldots, m \tag{2.5}$$

$$x_j \geq 0, \ \forall j \in J \tag{2.6}$$

Exemplo 2.10. A primeira coluna, por exemplo, corresponde a um padrão de corte com 2 rolos da largura 12, destinados ao primeiro cliente, e 1 rolo com a largura de 6, destinado ao terceiro cliente. Este padrão de corte não tem perdas, porque ocupa integralmente a largura do rolo. Considere-se que as quantidades pedidas são, respectivamente, 200, 300 e 100. Uma solução óptima seria o fornecimento de 100 rolos cortados segundo o padrão de corte 1, e 100 rolos segundo o padrão de corte 4.

larguras	padrões de corte							
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
12	2	1	1					$\geq$ 200
10		1		3	2	1		$\geq$ 300
6	1	1	3		1	3	5	$\geq$ 100
min	1	1	1	1	1	1	1	

Tabela 2.2: Quadro representando o modelo de corte

O ficheiro de input do LPSolve é o seguinte:

```
/* Função objectivo 1: minimizar o número de rolos a usar */
 min:  1 x1 + 1 x2 + 1 x3 + 1 x4 + 1 x5 + 1 x6 + 1x7;

/* Função objectivo 2: minimizar o desperdício existente nos rolos      */
/*                     satisfazendo a procura                           */
// min:          2 x2                  + 4 x5 + 2 x6 ;

/* Há situações em que as 2 funções objectivo conduzem à mesma solução */
/* Noutras situações, as soluções podem ser diferentes                 */

 2 x1 + x2 +   x3                                  >= 200;
        x2          + 3 x4 + 2 x5 + x6            >= 300;
   x1 + x2 + 3 x3          + 1 x5 + 3 x6 + 5 x7 >= 100;
```

□

Notas

Este modelo é devido a Gilmore e Gomory (1961 e 1963). Os autores mostraram que é suficiente usar padrões de corte maximais para encontrar a solução óptima fraccionária do problema. No entanto, a solução óptima inteira pode requerer o uso de padrões não maximais.

Dado que todas as combinações de itens que respeitam a largura dos rolos são possíveis, o número de padrões de corte é exponencialmente grande com respeito ao número de itens diferentes. Há técnicas especiais para resolver problemas deste tipo, designadas de geração diferidas de colunas, que permitem resolver em tempo razoável problemas de dimensão apreciável, com uma ou duas centenas de larguras diferentes de itens.

2.7 Problema de afectação generalizado

No problema de afectação generalizado, designado na literatura anglo-saxónica por *generalized assignment problem (GAP)*, pretende-se minimizar o custo de atribuição de um conjunto de itens a objectos de diferentes capacidade $W_i, \forall i$. Existe um peso associado a cada item j, w_{ij}, que depende do objecto i em que é colocado. Este problema é uma generalização do problema de empacotamento a uma dimensão. Existe uma formulação com padrões de empacotamento, mas vamos apresentar aqui uma formulação com variáveis de decisão com dois índices, x_{ij}, cujo significado é o seguinte:

$$x_{ij} = \begin{cases} 1 & \text{, se o item } j \text{ é atribuído ao bin } i \\ 0 & \text{, caso contrário} \end{cases}$$

O modelo é o seguinte:

$$\min z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{2.7}$$

$$\text{suj. a} \quad \sum_{j=1}^{n} w_{ij}x_{ij} \leq W_i,\ i = 1,\ldots,m \tag{2.8}$$

$$\sum_{i=1}^{m} x_{ij} = 1,\ j = 1,\ldots,n \tag{2.9}$$

$$x_{ij} = 0 \text{ ou } 1,\ \forall i,j \tag{2.10}$$

Exemplo 2.11. Este problema ocorre, por exemplo, no planeamento de operações de máquinas: os itens correspondem a tarefas que são atribuídas a máquinas, que são os objectos. O coeficiente w_{ij} indica o tempo de processamento da tarefa j na máquina i, W_i representa o tempo disponível na máquina i, e pretende-se minimizar o custo total de processamento, que é uma função dos coeficientes c_{ij}.

Considere os dados das seguintes tabelas que dizem respeito a um caso com 4 tarefas e 2 máquinas, cujas capacidades são 10 e 12, respectivamente. Os valores dos custos de processamento $C = [c_{ij}]$ são os seguintes:

$$C = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 15 & 12 & 10 & 9 \\ 2 & 8 & 4 & 8 & 6 \end{array}$$

Os valores dos tempos de processamento $P = [p_{ij}]$ são os seguintee:

$$P = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 3 & 4 & 5 & 6 \\ 2 & 4 & 5 & 8 & 4 \end{array}$$

O modelo de programação linear é o seguinte:

```
/* função objectivo */
min:     15 x11 + 12 x12 + 10 x13 +  9 x14 +
          8 x21 +  4 x22 +  8 x23 +  6 x24 ;

/* Restricões */
machine1:  3 x11 +  4 x12 +  5 x13 +  6 x14 <= 10;
machine2:  4 x21 +  5 x22 +  8 x23 +  4 x24 <= 12;

job1: x11 + x21 = 1;
job2: x12 + x22 = 1;
job3: x13 + x23 = 1;
job4: x14 + x24 = 1;

/* Restricões de integralidade */
bin x11, x12, x13, x14;
bin x21, x22, x23, x24;
```

A solução óptima deste exemplo, fornecida pelo LPSolve, significa que se devem atribuir as tarefas 1 e 3 à máquina 1 e as tarefas 2 e 4 à máquina 2, sendo o custo óptimo igual a 35.

Variables	MILP Feasible	MILP Better	MILP Better	result
	37	36	35	35
x11	0	0	1	1
x12	1	1	0	0
x13	0	1	1	1
x14	1	0	0	0
x21	1	1	0	0
x22	0	0	1	1
x23	1	0	0	0
x24	0	1	1	1

□

2.8 Lotes de produção

O objectivo e determinar a dimensão dos lotes a fabricar em cada período, dentro de um horizonte de planeamento, de modo a minimizar a soma dos custos de produção e dos custos de armazenagem, satisfazendo a procura em cada período.

Há situações em que se justifica a criação de inventário, como, por exemplo, quando há necessidade de dar resposta, em determinados períodos, a uma procura elevada (que até pode ser maior do que a capacidade de produção) e quando há benefícios de produzir em períodos em que os custos de produção são menores.

Em cada período, se o número de unidades disponíveis (*i.e.*, as unidades produzidas no período mais as existentes em stock) for superior à procura nesse período, as unidades remanescentes podem ser armazenadas em stock para venda em períodos subsequentes. A parte de cima do Figura 2.3 mostra a evolução do nível de stock (*i.e.*, o número de unidades em stock). Durante o primeiro período, o stock sobe, desde um valor inicial (s_0) até ao valor final (s_1).

O segmento de recta que une os pontos que representam os níveis de stock é uma aproximação da realidade que traduz a diferença entre a produção e o consumo num dado período. Normalmente, a produção ocorre continuamente ao longo do período, enquanto que a entrega das unidades ao cliente pode ocorrer de uma única vez ou ser dividida em parcelas ao longo do período.

O declive do segmento de recta indica a relação entre a taxa de produção e a de consumo. No primeiro período, a taxa de produção de artigos é maior do que a taxa de consumo; notar que o declive do segmento de recta é igual à diferença entre a taxa de produção e a de consumo. No último período, passa-se o oposto.

A parte de baixo da Figura 2.3 mostra uma representação esquemática das variáveis de decisão relevantes do sistema produtivo. As variáveis x_j designam o número de unidades produzidas no período j e as variáveis s_j o stock existente após o período j, para $j = 1,\ldots,T$. Os valores do número de unidades em stock no final dos períodos são usados para estimar o valor médio do número de unidades em stock, que é usado para determinar os custos de inventário.

Neste problema são dados:

- T : número de períodos do horizonte de planeamento
- d_j : procura existente no período j , $j = 1,\ldots,T$
- c_j : custo unitário de produção dos artigos no período j , $j = 1,\ldots,T$

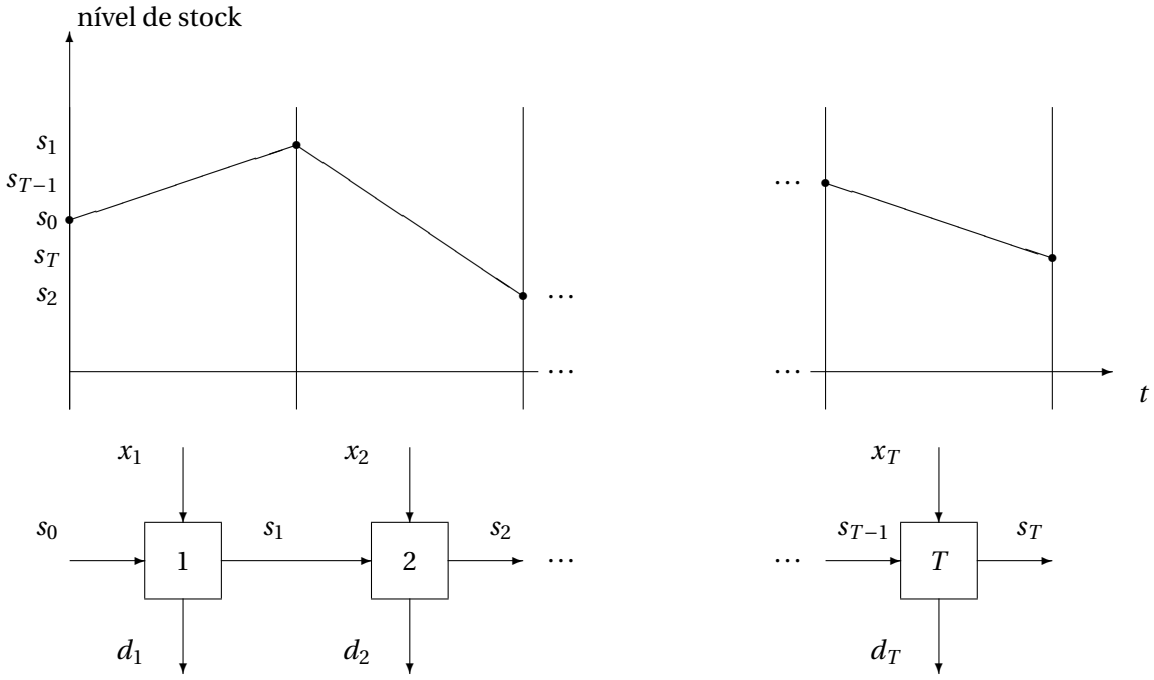


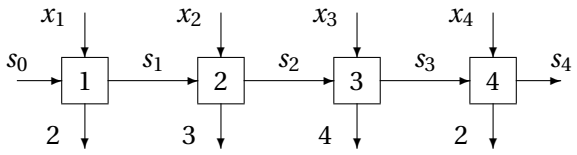
Figura 2.3: Evolução do stock ao longo do tempo e representação do modelo

- h_j : custo unitário de posse de inventário no período j , $j = 1,\ldots,T$
- x_j^{max} : número máximo de unidades produzidas no período j , $j = 1,\ldots,T$
- s_j^{max} : nível máximo de stock no período j , $j = 1,\ldots,T$
- s_0 e s_n : stocks inicial e final, respectivamente

O modelo geral de programação linear é o seguinte.

$$\begin{aligned} \min \quad & \sum_{j=1}^{T} (c_j x_j + h_j s_j) \\ \text{suj. a} \quad & x_j + s_{j-1} - s_j = d_j \ , \ j = 1,\ldots,T \\ & 0 \le x_j \le x_j^{max} \ , \ j = 1,\ldots,T \\ & 0 \le s_j \le s_j^{max} \ , \ j = 1,\ldots,T \end{aligned}$$

Exemplo 2.12. Considere o exemplo com um horizonte de planeamento (T) de 4 períodos, correspondendo ao seguinte esquema:



com os seguintes dados:

- Procura em cada período de 2, 3, 4 e 2, respectivamente.
- Capacidade máxima de produção, x_j^{max} : 4 unidades em cada período.
- Nível máximo de stock, s_{max} : 2 unidades.
- Custos unitários de armazenagem, h_j : 1 U.M./ artigo x período.
- Custos unitários de produção: custo variável proporcional ao número de artigos, p_j, listados na seguinte tabela:

j	1	2	3	4
p_j	12	10	14	10

O modelo de programação linear é o seguinte:

```
/* Função objectivo */
min: 12 x1 + 10 x2 + 14 x3 + 10 x4 + 1 s1 + 1 s2 + 1 s3 + 1 s4;

/* Restrições */
s0 = 0;
x1 + s0 - s1 = 2;
x2 + s1 - s2 = 3;
x3 + s2 - s3 = 4;
x4 + s3 - s4 = 2;
s4 = 0;

// Capacidade máxima de produção: 4 unidades / período
x1 <= 4; x2 <= 4; x3 <= 4; x4 <= 4;

// Inventário máximo:  2 unidades
s1 <= 2; s2 <= 2; s3 <= 2; s4 <= 2;
```

A solução óptima do problema, com um custo de 127, é $x_1 = 3, x_2 = 4, x_3 = 2$ e $x_4 = 2$. Os valores dos stocks são $s_1 = 1, s_2 = 2$ e $s_0 = s_3 = s_4 = 0$.

□

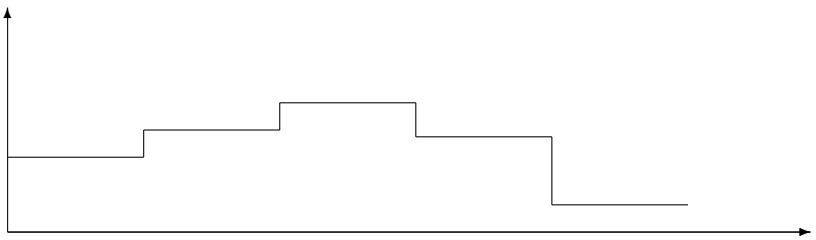
Esta formulação pode ser estendida para situações multi-artigo, em que as restrições de capacidade de produção e de inventário dizem respeito à totalidade dos artigos produzidos (ver Aplicações)

2.9 Problema de gestão de pessoal

Há situações em que é necessário estabelecer uma política de contratações para satisfazer necessidades de pessoal que variam ao longo do tempo. Isso ocorre no planeamento de pessoal em serviços de funcionamento diário permanente ao longo das 24 horas (*e.g.*, hospitais, *call centers*, portagens em auto-estradas) ou num horizonte de planeamento com um conjunto de períodos (*e.g.*, uma obra que decorre durante vários meses).

No primeiro caso, pode haver durações alternativas para o tempo de serviço prestado na jornada de trabalho, que podem também obedecer a outras especificações. No segundo caso, pode haver contratos por durações pré-determinadas e eventualmente custos de contratação, treino e despedimento de pessoal com contratos a termo certo.

Exemplo 2.13. A seguinte figura mostra as necessidades de pessoal ao longo do tempo:



□

Uma forma de modelar estes problemas é discretizando o tempo, havendo uma restrição associada a cada um dos períodos de tempo. Cada variável de decisão (coluna) corresponde a uma acção de contratação permitida que cobre um conjunto de períodos. A coluna tem elementos iguais a 1 nos períodos em que a respectiva contratação presta serviço e elemento nulos nos restantes períodos.

Seja m o número de períodos. O modelo usa variáveis de decisão x_{ij} que designam o número de trabalhadores contratados desde o início do período i até ao fim do período $j, 1 \leq i \leq j \leq m$. A coluna da variável x_{ij} tem elementos iguais a 1 desde o período i até ao período j, e nulos nos restantes períodos. O coeficiente de custo c_{ij} é o custo de contratação, treino e despedimento de um trabalhador com contrato desde o início do período i até ao fim do período j. e ordenados pagos durante esse período.

Exemplo 2.14. Vamos considerar um modelo do segundo caso, em que se pretende planear as contratações para um horizonte temporal de vários meses. Para um horizonte de 5 meses e as necessidades de pessoal indicadas no lado direito, o seguinte quadro apresenta os diversos tipos de contratação permitidos. Os valores de custos c_{ij} foram calculados usando um ordenado mensal igual a 1 U.M./mês e um custo de contratação, treino e despedimento igual a 1 U.M.. O quadro que representa o modelo é:

	x_{15}	x_{13}	x_{24}	x_{35}	x_{12}	x_{23}	x_{34}	x_{22}		
1	1	1			1				$\geq$	6
2	1	1	1		1	1		1		10
3	1	1	1	1		1	1			14
4	1		1	1			1			9
5	1			1						8
c_{ij}	6	4	4	4	3	3	3	2		
x^*	4	2	1	4	0	3	0	0		

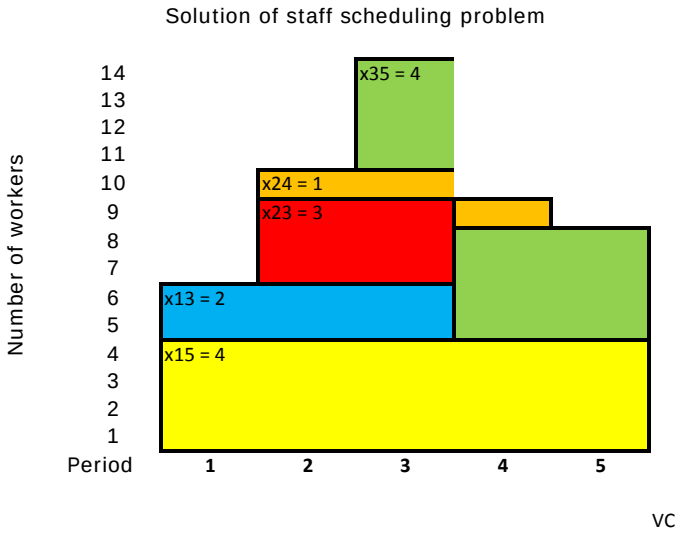
O ficheiro de *input* do LPSolve é o seguinte:

```
/* função objectivo */
min: 6 x15 + 4 x13 + 4 x24 + 4 x35 + 3 x12 + 3 x23 + 3 x34 + 2 x22;

/* restrições */

x15 + x13 + x12 >= 6;
x15 + x13 + x24 + x12 + x23 + x22 >= 10;
x15 + x13 + x24 + x35 + x23 + x34 >= 14;
x15 + x24 + x35 + x34 >= 9;
x15 + x35 >= 8;
```

O seguinte quadro mostra a solução óptima. É a solução x^*, que tem um custo igual a 61 unidades.

Variables	result
	61
x15	4
x13	2
x24	1
x35	4
x12	0
x23	3
x34	0
x22	0

A seguinte figura mostra o número de trabalhadores contratados em cada tipo de contrato na solução óptima, sendo cada tipo de contrato representado a uma cor diferente.

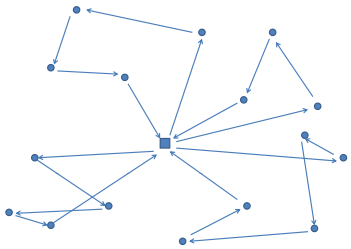


□

Neste exemplo, o número de trabalhadores em cada período é igual ao requerido, mas pode haver situações em que a solução mais económica tem excesso de trabalhadores num período ou até em mais períodos. Tal ocorre, por exemplo, se os custos de contratação forem elevados, em que é preferível incorrer num custo de não-utilização, em vez de suportar maiores custos de contratação.

2.10 Planeamento de rotas de veículos

Os problemas de planeamento de rotas de veículos consistem em determinar o conjunto de itinerários a seguir pelos veículos de uma frota para visitar um dado conjunto de clientes. O itinerário de cada veículo tem início no depósito, percorre os trajectos entre os clientes, e regressa ao depósito, como se ilustra na Figura.



Uma das restrições comuns à generalidade do problemas é a relativa à capacidade do veículo. As cargas dos clientes seleccionados para um itinerário devem poder ser acondicionadas no veículo. Nalguns problemas de planeamento de rotas de veículos, é também necessário ter em linha de conta restrições que não permitem que os serviços sejam efectuados fora de intervalos de

tempo previamente definidos. Estas restrições podem resultar de condicionamentos de tráfego de veículos de grandes porte em zonas urbanas durante alguns períodos, ou de restrições organizativas que impõem um calendário para efectuar a descarga das mercadorias.

Problemas que envolvem estas restrições são designados por problemas de planeamento de rotas com janelas temporais. Uma janela temporal define o intervalo de tempo em que é permitido iniciar o serviço no cliente, *i.e.*, o veículo não pode chegar ao cliente depois de terminar a janela temporal, e, se chegar antes, deve esperar até o instante em que ela começa.

Os problemas de rotas de veículos podem envolver todos os custos relacionados com a operação da frota, nomeadamente os custos fixos de utilização dos veículos e os custos variáveis relacionados com as distâncias e os tempos de viagem, os custos de carga e de descarga e os custos de tempos de espera.

Estes problemas têm sido abordados com sucesso recorrendo a formulações em que cada variável de decisão corresponde a um trajecto possível de um veículo. Estas variáveis de decisão são binárias, traduzindo a selecção, ou não selecção, de cada trajecto. A coluna da variável de decisão guarda informação sobre os clientes visitados, e o coeficiente de custo associado à variável de decisão pode incluir todos os custos relativos ao trajecto, nomeadamente os custos fixos e os variáveis. O objectivo do problema é escolher o subconjunto de trajectos que realizam todos os serviços com um custo mínimo.

Seja P o conjunto de caminhos possíveis, cada um deles satisfazendo todas as restrições impostas ao problema. Seja y_p uma variável binária que indica se o caminho $p \in P$ é usado, ou não, c_p o custo de utilizar o caminho p, e

$$\delta_{ip} = \begin{cases} 1 & \text{, se o caminho } p \text{ visita o cliente } i \\ 0 & \text{, caso contrário} \end{cases}$$

O modelo de planeamento de rotas é o seguinte:

$$\min \quad \sum_{p \in P} c_p y_p \tag{2.11}$$

$$\text{subj. to} \quad \sum_{p \in P} \delta_{ip} y_p = 1 \ , \ \forall i \in V \tag{2.12}$$

$$y_p \text{ binário} \ , \ \forall p \in P \tag{2.13}$$

Para definir cada coluna, é necessário resolver um problema num grafo auxiliar em que se estabelece quais os clientes j que é possível visitar depois de ter visitado o cliente i. Essa informação pode ser representada num grafo em que existe um arco entre o vértice i e o vértice j, se tal sequência for possível. O grafo pode ser completado adicionando um vértice correspondendo ao depósito, que deve funcionar como início e fim da viagem de cada veículo. É de salientar que este tipo de formulação permite uma grande flexibilidade de modelação de diferentes tipos de problemas.

Exemplo 2.15. Considere um problema de reduzidas dimensões em que é necessário visitar 8 clientes localizados em pontos distintos. Foi feita uma enumeração de todos os trajectos que podiam ser efectuados pelos veículos, de modo a respeitar as janelas temporais de visita aos clientes, entrando em consideração com os tempos de viagem entre os clientes, e de e para o depósito. Foi estabelecido que havia 15 possibilidades distintas. Esses trajectos estão representados na Figura 2.4,

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}		
cliente 1	1	1	1	1	1	1										=	1
2	1	1					1	1	1	1	1					=	1
3			1	1			1					1	1	1		=	1
4	1							1	1			1				=	1
5					1					1					1	=	1
6		1	1					1					1			=	1
7				1			1				1	1		1		=	1
8						1					1		1	1	1	=	1
custo	8	7	10	9	8	7	10	11	7	6	10	9	10	12	7		

Figura 2.4: Cobertura de viagens com trajectos

cada um deles associado a uma coluna. A título de exemplo, o trajecto que corresponde à variável x_{15} permite visitar os clientes 5 e 8, com um custo de 7 U.M..

O problema de determinar o conjunto de trajectos a seleccionar para visitar todos os clientes a um custo mínimo corresponde a determinar a solução óptima do problema de programação inteira, com variáveis binárias, definido no Quadro.

É de salientar que apenas foram enumerados os trajectos maximais (um trajecto não é maximal se puder ainda visitar mais algum cliente), embora a solução óptima possa ter trajectos que não são desse tipo. □

Neste tipo de modelos, para problemas de pequena dimensão, é possível proceder a uma enumeração completa de todas as alternativas, e resolver o correspondente problema de programação inteira binária. Quando a dimensão do problema é demasiado grande, as dificuldades relacionadas com o grande número de trajectos possíveis podem ser ultrapassadas recorrendo ao método da geração diferida de colunas, em que não é necessário recorrer à enumeração explícita de todas as colunas do problema.

2.11 Transformações básicas

2.11.1 Inequação do tipo ≤ numa equação

Qualquer inequação do tipo de menor ou igual pode ser transformada numa equação, introduzindo uma variável adicional com valor não-negativo:

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i \qquad \Leftrightarrow \qquad \sum_{j=1}^{n} a_{ij}x_j + s_i = b_i, s_i \geq 0.$$

A variável s_i é designada por variável de folga, pela razão indicada de seguida.

Exemplo 2.16. Considere-se a restrição $2x_1 + 3x_2 + 4x_4 \leq 8$, relativa ao uso de um dado recurso, cuja quantidade disponível é 8. A função linear $2x_1 - 3x_2 + 4x_4$ indica a quantidade de recurso usada em função dos valores das variáveis de decisão.

$$\begin{aligned} 2x_1 + 3x_2 + 4x_4 \leq 8 \qquad &\Leftrightarrow \qquad 2x_1 + 3x_2 + 4x_4 + s_1 = 8 \\ &\qquad\quad s_1 \geq 0 \end{aligned}$$

A variável s_1 indica a quantidade de recurso não usada, *i.e.*, $s_1 = 8 - 2x_1 - 3x_2 - 4x_4$, e portanto representa a folga do recurso. □

2.11.2 Inequação do tipo ≥ numa equação

Qualquer inequação do tipo de maior ou igual pode ser transformada numa equação, introduzindo uma variável adicional com valor não-negativo:

$$\sum_{j=1}^{n} a_{ij}x_j \geq b_i \qquad \Leftrightarrow \qquad \sum_{j=1}^{n} a_{ij}x_j - s_i = b_i, s_i \geq 0.$$

A variável s_i é designada por variável de excesso, pela razão indicada de seguida. Alguns autores também a designam por variável de folga.

Exemplo 2.17. Considere-se a restrição $2x_1 + 3x_2 + 4x_4 \geq 4$, relativa à quantidade produzida de um dado bem ou mercadoria, cuja quantidade necessária é 4. A função linear $2x_1 + 3x_2 + 4x_4$ indica a quantidade produzida em função dos valores das variáveis de decisão.

$$\begin{aligned} 2x_1 + 3x_2 + 4x_4 \geq 4 \qquad \Leftrightarrow \qquad & 2x_1 + 3x_2 + 4x_4 - s_1 = 4 \\ & s_1 \geq 0 \end{aligned}$$

A variável s_1 indica o excesso de produção em relação à quantidade requerida, *i.e.*, $s_1 = 1x_1 + 2x_2 + 3x_4 - 4$, e portanto representa o excesso de produção. □

2.11.3 Equação em duas inequações

Qualquer restrição de igualdade pode ser expressa como uma par de inequações do tipo de menor ou igual:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \qquad \Leftrightarrow \qquad \left\{ \begin{array}{rcl} \sum_{j=1}^{n} a_{ij}x_j & \leq & b_i \\ \sum_{j=1}^{n} a_{ij}x_j & \geq & b_i \end{array} \right. \qquad \left\{ \begin{array}{rcl} \sum_{j=1}^{n} a_{ij}x_j & \leq & b_i \\ -\sum_{j=1}^{n} a_{ij}x_j & \leq & -b_i \end{array} \right.$$

Exemplo 2.18.

Antes:	Depois:
$1x_1 - 2x_2 + 3x_4 = 4$	$1x_1 - 2x_2 + 3x_4 \leq 4$
	$1x_1 - 2x_2 + 3x_4 \geq 4$

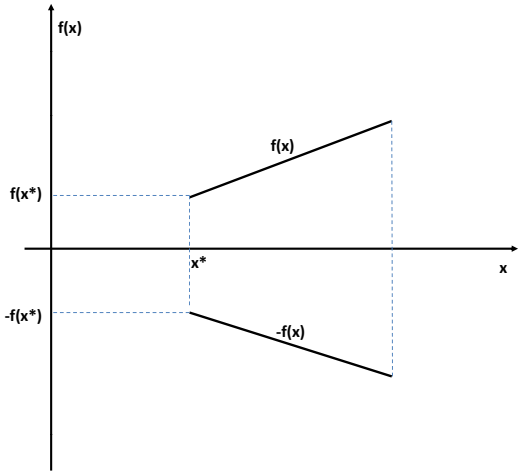
□

2.11.4 Problema de minimização em problema de maximização

Qualquer problema de minimização pode ser reduzido a um problema de maximização, em que se optimiza a função objectivo simétrica da original:

$$\min z = cx \Leftrightarrow \max z' = -cx.$$

A solução óptima x^* é a mesma, mas o valor da função objectivo da solução óptima é o simétrico $f(x^*) = \min f(x) = -\max -f(x)$



2.11.5 Variáveis sem restrição de sinal

Qualquer variável sem restrição de sinal pode ser expressa como a diferença de duas variáveis não-negativas:

$$x_j \text{ sem restrição} \Leftrightarrow x_j = x_j^+ - x_j^-, x_j^+ \geq 0, x_j^- \geq 0.$$

Exemplo 2.19. As restrições $2x_1 + 3x_2 \leq 20, x_1$ sem restrição, $x_2 \geq 0$ são transformadas fazendo a mudança de variável $x_1 = x_1^+ - x_1^-$, dando origem a $2x_1^+ - 2x_1^- + 3x_2 \leq 20,\ x_1^+, x_1^-, x_2 \geq 0$.

□

2.11.6 Variáveis com limite inferior

Uma variável com limite inferior pode ser substituída por uma variável com limite inferior igual a 0, por mudança de variável:

Exemplo 2.20. As restrições $2x_1 + 3x_2 \leq 20,\ x_1 \geq 8, x_2 \geq 0$ são transformadas fazendo a mudança de variável $x_1' = x_1 - 8 \rightarrow x_1 = x_1' + 8$, dando origem a $2(x_1' + 8) + 3x_2 \leq 20, x_1' \geq 0, x_2 \geq 0$, *i.e.*, $2x_1' + 3x_2 \leq 4, x_1' \geq 0, x_2 \geq 0$.

□

2.11.7 Restrições do tipo módulo (caso ≤)

- $$\left| \sum_{j=1}^{n} a_{ij} x_j \right| \leq b_i \Leftrightarrow \left\{ \begin{array}{rcl} \sum_{j=1}^n a_{ij} x_j & \leq & b_i \\ \sum_{j=1}^n a_{ij} x_j & \geq & -b_i \end{array} \right. \left\{ \begin{array}{rcl} \sum_{j=1}^n a_{ij} x_j & \leq & b_i \\ -\sum_{j=1}^n a_{ij} x_j & \leq & b_i \end{array} \right.$$

Exemplo 2.21.

- Antes: $|2x_1 + 3x_2| \leq 20$
- Depois: $\left\{ \begin{array}{rcl} 2x_1 + 3x_2 & \leq & 20 \\ 2x_1 + 3x_2 & \geq & -20 \end{array} \right.$
- Trata-se de uma conjunção de restrições.

□

2.11.8 Restrições do tipo módulo (caso ≥)

- $$\left|\sum_{j=1}^{n} a_{ij}x_j\right| \geq b_i \Leftrightarrow \left| \begin{array}{lcl} \sum_{j=1}^{n} a_{ij}x_j & \geq & b_i \\ \sum_{j=1}^{n} a_{ij}x_j & \leq & -b_i \end{array} \right.$$

Esta disjunção de condições não pode ser representada por uma conjunção de restrições lineares, porque uma conjunção de restrições lineares define sempre um domínio convexo.

Exemplo 2.22.

- $|x_1| \geq 2$
- equivale a: $\left| \begin{array}{lcl} x_1 & \leq & -2 \\ x_1 & \geq & 2 \end{array} \right.$
- Trata-se de um domínio não-convexo.

□

Capítulo 3

Fluxos em rede

Há muitos problemas que podem ser modelados como fluxos em rede. Estes problemas têm uma estrutura especial. As matrizes de incidência arco-vértice são totalmente unimodulares. Em matrizes com esta propriedade, qualquer base (conjunto de vectores lineares independentes) tem um determinante igual a $+1$ ou -1. Como consequência disso, dado que a matriz inversa é a adjunta da transposta dividida pelo determinante, todos os elementos da matriz B^{-1} são inteiros.

Nesta circunstância, se os valores das ofertas e dos consumos forem inteiros, todas as soluções básicas são soluções inteiras. Assim, mesmo que as restrições de integralidade sejam substituídas por restrições de não–negatividade ($x_{ij} \geq 0$), a solução óptima é sempre inteira.

Mas não é habitual usar *packages* de programação linear para resolver estes problemas de fluxos em rede, também designados por problemas de transporte em rede com limites de capacidade nos arcos. A propriedade que garante que todas as soluções básicas são inteiras e a estrutura especial da matriz, apenas com 2 elementos diferentes de 0, iguais a $+1$ ou -1, permite desenvolver algoritmos especializados mais eficientes, que podem ser uma centena de vezes mais rápidos do que um *package* de programação linear.

Este capítulo inicia-se com o modelo geral de transporte em rede e uma propriedade estrutural das soluções (ver Teorema 1. Conhecer a estrutura das soluções de um problema é interessante por si só, mas adicionalmente pode ajudar a conceber modelos diferentes.

Depois, apresentam-se modelos de programação linear de vários problemas típicos de optimização que podem ser vistos como casos particulares do problema de transporte em rede. Do ponto de vista prático, não é habitual recorrer-se a programação linear para resolver alguns dos problemas apresentados de seguida, como, por exemplo, o problema do caminho mais curto. Para esse problema, existem algoritmos combinatórios especializados, que são muito mais eficientes. São exemplos os algoritmos de Dijkstra, Bellman-Ford e Floyd.

Isso não é razão para os modelos de programação linear deixarem de ter utilidade. Muitas vezes, os modelos que vamos analisar são uma componente dos modelos de um problema mais complexo que estamos interessados em resolver. A título de exemplo, o problema do caminho mais curto com uma restrição do tempo de viagem. Neste problema, existem dois parâmetros associados a cada arco, a distância e o tempo de viagem, e o objectivo é determinar o caminho mais curto cujo tempo de viagem não exceda um tempo pré-fixado. É um problema para o qual não é conhecido nenhum algoritmo polinomial, que requer o uso de programação inteira.

Há também outros problemas que podem ser representados como problemas de fluxo em rede com restrições adicionais. As soluções, além de obedecerem à propriedade estrutura de serem um fluxo numa rede, também devem respeitar um conjunto de restrições adicionais. São apresenta-

dos os exemplos de um modelo de corte / empacotamento a uma dimensão e de um modelo do problema do caixeiro viajante. As restrições adicionais destroem a propriedade de total unimodularidade, e os problemas de transporte com restrições adicionais já não podem ser resolvidos com software de resolução de problemas em rede. Estes dois problemas são NP-completos. É necessário recorrer a software de resolução de problemas de programação inteira, como, por exemplo, o LPSolve.

Para além dos problemas em rede, apresentam-se também neste capítulo dois problemas que são problemas duais do problema do caminho mais curto e do problema de fluxo máximo. São eles o problema do máximo afastamento de vértices e o problema do corte mínimo, respectivamente.

3.1 Modelo geral

Dado um grafo $G = (V, A)$, em que V é conjunto de vértices e A o conjunto de arcos, e sendo dados c_{ij}, o custo unitário de transporte no arco orientado $(i, j) \in A$, u_{ij}, a capacidade do arco orientado $(i, j) \in A$ e b_j, o valor da oferta (valor positivo) ou procura (valor negativo) no vértice $j \in V$, pretende-se determinar o fluxo admissível de custo mínimo, representado por um conjunto de variáveis de decisão x_{ij}, o fluxo de *um único tipo de entidades* no arco orientado $(i, j) \in A$.

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} \\ \text{suj. a} \quad & -\sum_{(i,j)\in A} x_{ij} + \sum_{(j,i)\in A} x_{ji} = b_j, \ \forall j \in V \qquad (3.1) \\ & 0 \le x_{ij} \le u_{ij}, \ \forall (i,j) \in A \qquad (3.2) \end{aligned}$$

As restrições (3.1), designadas por *restrições de conservação de fluxo*, forçam o fluxo que entra num vértice a ser igual ao fluxo que sai (ofertas e procuras incluídas); as restrições (3.2), designadas por *restrições de capacidade,* não permitem que o fluxo num dado arco exceda a sua capacidade.

As soluções de um problema de fluxo em rede têm uma estrutura especial. Sendo $|V| = n$ e $|A| = m$, qualquer solução (conjunto de valores de fluxo não-negativo nos arcos da rede) obedece a:

Teorema 1 (Teorema da decomposição de fluxos (Ahuja et al.,93) [1])**.** *Um fluxo não-negativo numa rede pode ser representado como um conjunto de fluxos em caminhos e em ciclos (não necessariamente de uma forma única) com as seguintes duas propriedades:*

(a) cada caminho com fluxo positivo liga um vértice de oferta a um vértice de consumo.

(b) no máximo $n + m$ caminhos e ciclos têm fluxo positivo; destes, no máximo, m ciclos têm fluxo positivo.

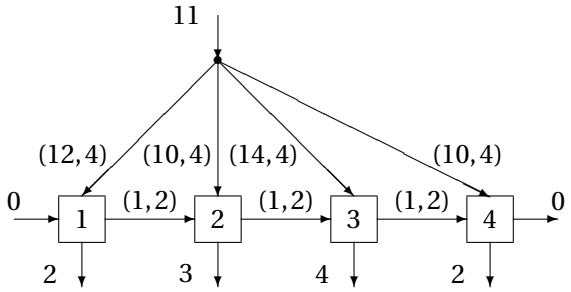
Inversamente, um dado conjunto de fluxos em caminhos e em ciclos tem uma representação única como um fluxo não-negativo numa rede.

Apresenta-se apenas um esboço da prova: podemos iterativamente identificar um caminho e retirar fluxo ao longo desse caminho entre um vértice de oferta e um vértice de consumo. A operação é válida, dado haver conservação de fluxo nos vértices. Se, no final restar fluxo, deve ser em ciclos; de novo, pelas restrições de conservação de fluxo, podemos repetir a operação. Os números máximos de caminhos e ciclos resultam do número de restrições do modelo.

3.2 Lotes de produção

O problema dos lotes de produção pode ser formalizado com uma problema de transporte em rede geral com capacidades associadas aos arcos. Os períodos correspondem a vértices e as variáveis de decisão associadas à produção e ao stock correspondem a arcos. Adicionalmente, há um vértice fictício, que agrega as origens de todos os arcos correspondentes a produção. O valor indicado para a oferta desse vértice foi escolhido para tornar o problema balanceado (soma das ofertas = soma dos procuras).

Exemplo 3.1. A rede associada ao exemplo apresentado no capítulo anterior é mostrada na Figura. Trata-se de uma rede com capacidades associadas aos arcos. Os valores associados aos arcos, (c_{ij}, u_{ij}), representam o custo unitário de transporte e a capacidade do arco, respectivamente, e os valores associados aos vértices representam ofertas e procuras.



Sendo o vértice fictício (de origem dos arcos de produção) o vértice 5, o ficheiro de *input* do Relax4 para este exemplo é:

```
5
7
5 1 12 4
5 2 10 4
5 3 14 4
5 4 10 4
1 2  1 2
2 3  1 2
3 4  1 2
-2
-3
-4
-2
11
```

O output do Relax4, mostrado na janela de DOS (con:), é o seguinte:

```
C:\Desktop\RELAX4>relax4 <LotesProducao.txt >con:
 END OF READING
 NUMBER OF NODES = 5, NUMBER OF ARCS = 7
 CONSTRUCT LINKED LISTS FOR THE PROBLEM
 CALLING RELAX4 TO SOLVE THE PROBLEM
```

```
**********************************
TOTAL SOLUTION TIME =  0. SECS.
TIME IN INITIALIZATION =  0. SECS.
  5 1  2.
  5 2  4.
  5 3  3.
  5 4  2.
  2 3  1.
OPTIMAL COST =   127.
NUMBER OF AUCTION/SHORTEST PATH ITERATIONS = 9
NUMBER OF ITERATIONS =  7
NUMBER OF MULTINODE ITERATIONS =  2
NUMBER OF MULTINODE ASCENT STEPS =  0
NUMBER OF REGULAR AUGMENTATIONS =  2
**********************************
```

Trata-se de uma solução óptima alternativa da encontrada com o package de programação linear.

□

3.3 Problema do caminho mais curto

Um dos problemas com maior aplicação em grafos é o problema do caminho mais curto. Pretende-se determinar o caminho mais curto ou mais rápido entre dois ou mais pontos de uma rede; entre as inúmeras aplicações, citam-se a concepção de redes de comunicação e os problemas de transporte e de distribuição.

Considere um grafo $G = (V, A)$, sendo V o conjunto de vértices e A o conjunto de arcos orientados. Cada arco é designado como (i, j), sendo i o vértice de origem do arco e j o vértice de destino. A cada arco está associado um custo c_{ij}.

Pretende-se determinar o caminho mais curto entre dois vértices do grafo, designados por s e t. Um caminho é formado por uma sequência de arcos, e cada solução admissível é uma sequência de arcos entre os vértices s e t. O problema é formulado injectando no grafo, no vértice s, uma unidade de fluxo que passa por uma sequência de arcos até atingir o vértice t. As restrições impõem que, se a unidade entrar num vértice (excepto s e t), deve também sair dele. A função objectivo deve incluir termos correspondentes aos custos dos arcos, tomando como valor a soma dos custos dos arcos seleccionados. A minimização de custo determina a escolha do caminho mais curto.

São dados do problema os custos c_{ij} associados ao arco $(i, j), \forall j$. As variáveis de decisão x_{ij} associadas a cada arco (i, j) do grafo tomam o valor 1, se o arco fizer parte do caminho mais curto, e o valor 0, caso contrário. O modelo é o seguinte:

$$\min \quad \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{3.3}$$

$$\text{suj. a} \quad -\sum_{(i,j)\in A} x_{ij} + \sum_{(j,k)\in A} x_{jk} = \begin{cases} 1 & , \text{ se } j = s \\ 0 & , \text{ se } j \neq s, t \\ -1 & , \text{ se } j = t \end{cases} \tag{3.4}$$

$$x_{ij} \text{ binário }, \ \forall (i, j) \in A \tag{3.5}$$

As restrições do problema, relativas a cada um dos vértices do grafo, traduzem a conservação de fluxo em cada vértice, ou seja, o número de unidades de fluxo que entram no vértice j através dos arcos (i, j) deve ser igual ao número de unidades que dele saem através dos arcos (j, k). Esta relação é válida para todos os vértices, excepção feita ao vértice s, onde é introduzida uma unidade de fluxo na rede, e ao vértice t, onde a unidade é removida.

Exemplo 3.2. Considere a rede apresentada na Figura 3.1, em que os custos associados aos arcos, que apenas podem ser percorridos no sentido indicado, representam as distâncias entre localidades, representadas pelos vértices.

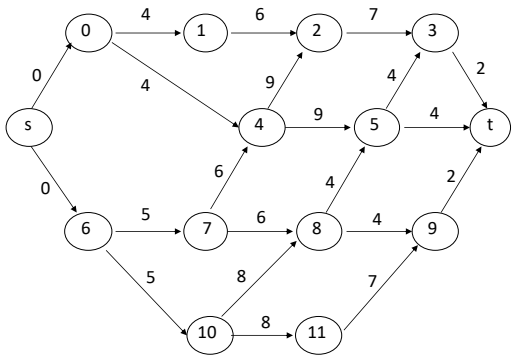


Figura 3.1: Rede

Para o Exemplo apresentado na Figura 3.1, estas restrições são as apresentadas na Tabela 3.1.

	x_{s0}	x_{s6}	x_{01}	x_{04}	x_{12}	x_{23}	x_{3t}	x_{42}	x_{45}	x_{53}	x_{5t}	x_{610}	x_{67}	x_{74}	x_{78}	x_{85}	x_{89}	x_{9t}	x_{1011}	x_{108}	x_{119}	
vértice s	1	1																				1
vértice 0	-1		1	1																		0
vértice 1			-1		1																	0
vértice 2					-1	1		-1														0
vértice 3						-1	1			-1												0
vértice 4				-1				1	1					-1								0
vértice 5									-1	1	1					-1						0
vértice 6		-1										1	1									0
vértice 7													-1	1	1							0
vértice 8															-1	1	1			-1		0
vértice 9																	-1	1			-1	0
vértice 10																			1	1		0
vértice 11												-1							-1		1	0
vértice t							-1				-1							-1				-1
min	0	0	4	4	6	7	2	9	9	4	4	5	5	6	6	4	4	2	8	8	2	

Tabela 3.1: Modelo do problema de caminho mais curto

Este problema pode ser resolvido com software de optimização de redes. Para depois mostrar as relações entre este problema e o problema de afastamento máximo de um vértice, apresenta-se de seguida o ficheiro de input do LPSolve.

```
//   caminho mais curto num grafo acíclico
min: 4 x01 + 4 x04 + 6 x12 + 7 x23 + 2 x3t + 9 x42 + 9 x45 + 4 x53 +
      4 x5t + 5 x67 + 5 x610 + 6 x74 + 6 x78 + 4 x85 + 4 x89 + 2 x9t +
      8 x108 + 8 x1011 + 7 x119;

/* Restrições */
// fluxo que entra no vértice = fluxo que sai
vertice_s: xs0 + xs6 = 1;
vertice_0: xs0 = x01 + x04;
vertice_1: x01 = x12;
vertice_2: x12 + x42 = x23;
vertice_3: x23 + x53 = x3t;
vertice_4: x04 + x74 = x42 + x45;
vertice_5: x45 + x85 = x53 + x5t;
vertice_6: xs6 = x67 + x610;
vertice_7: x67 = x74 + x78;
vertice_8: x78 + x108 = x85 + x89;
vertice_9: x89 + x119 = x9t;
vertice_10: x610 = x108 + x1011;
vertice_11: x1011 = x119;
```

As variáveis de decisão que tomam o valor 1 na solução óptima definem o caminho mais curto entre os vértices s e t. No problema em análise, é o caminho constituído pela sequência de vértices $s, 6, 7, 8, 9, t$ e pelos arcos que os unem, com uma distância de 17 unidades.

Esta distância é igual à distância máxima a que se pode colocar o vértice t em relação ao vértice s no problema de afastamento máximo dos vértices, tendo em conta que dois quaisquer vértices não podem ter um afastamento maior do que o comprimento do arco que os une (ver Secção 3.11). □

Notas

A determinação do caminho mais curto num grafo pode ser determinada em tempo polinomial, e existem algoritmos combinatórios especializados para resolver este problema, nomeadamente o algoritmo de Dijkstra, para situações em que não existem custos negativos, e o algoritmo de Ford, para situações gerais. O algoritmo de Floyd pode ser usado para determinar as distâncias mais curtas entre cada par de vértices de um grafo.

3.4 Problema do fluxo máximo

O problema de fluxo máximo permite avaliar a capacidade de transporte de um bem ou mercadoria desde centros de produção até centros de consumo em redes com limites de capacidade nos arcos.

Seja uma rede $R = (V, A, s, t)$ definida sobre um grafo $G = (V, A)$, sendo V o conjunto de vértices e A o conjunto de arcos orientados, com dois vértices distintos, a fonte e o terminal, designados por s e t. Considera-se o vértice s como sendo uma fonte ilimitada de fluxo e o vértice t como um terminal capaz de absorver todo o fluxo conduzido pelos outros arcos. Nos restantes vértices do grafo, existe conservação de fluxo, *i.e.*, o fluxo que chega ao vértice deve ser igual ao fluxo que dele

sai. A cada arco, está associada uma capacidade máxima que corresponde ao maior fluxo possível que pode circular no arco. O objectivo do problema consiste em determinar o máximo fluxo que pode circular entre a fonte e o terminal, sujeito às restrições de capacidade nos arcos.

O problema de fluxo máximo pode ser formulado como um problema de programação linear com arcos com limite superior. O fluxo em cada arco é designado por x_{ij} e a respectiva capacidade máxima por l_{ij}. O fluxo no grafo, que designaremos por f, pode ser expresso como a quantidade de fluxo que sai no vértice s ou como a quantidade de fluxo que chega ao vértice t. Por exemplo:

$$f - \sum_{(s,j)\in A} x_{sj} \qquad = 0$$

O objectivo do problema é determinar a solução válida de maior fluxo.

$$\begin{array}{ll} \max & f \\ \text{suj. a} & -\sum_{(i,j)\in A} x_{ij} + \sum_{(j,k)\in A} x_{jk} = \begin{cases} f & , \text{ se } j = s \\ 0 & , \text{ se } j \neq s,t \\ -f & , \text{ se } j = t \end{cases} \\ & 0 \leq x_{ij} \leq l_{ij} \ , \ \forall (i,j) \in A \end{array}$$

O primeiro conjunto de condições estabelece a capacidade máxima de cada arco. O segundo conjunto de condições diz respeito à conservação de fluxo nos vértices do digrafo.

Um arco tem fluxo nulo, se $x_{ij} = 0$. Por outro lado, se $x_{ij} = l_{ij}$, diz–se que o arco está saturado. Para redes sem limites inferiores de fluxo, existe trivialmente um fluxo válido, com todos os $x_{ij} = 0$.

Exemplo 3.3. A Figura 3.2 apresenta uma rede de transporte desde os campos de petróleo localizados no ponto s até à refinaria localizada no ponto t. Cada troço do oleoduto tem a capacidade [Mlitros/hora] indicada junto do respectivo arco. O transporte apenas pode ser efectuado no sentido de bombagem indicado pelas setas. A capacidade máxima de transporte do oleoduto é dada pelo fluxo máximo entre os pontos s e t.

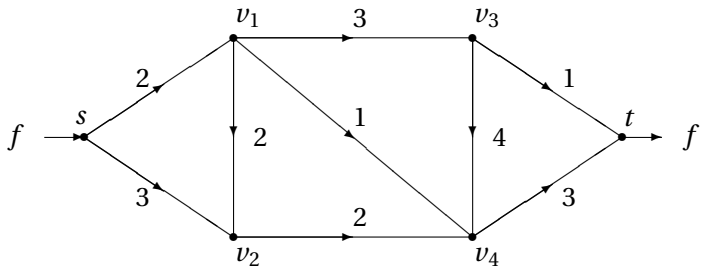


Figura 3.2: Rede

Para a rede apresentada na Figura 3.2, estas restrições são as apresentadas no Quadro 3.2.
O ficheiro de input do LPSolve é:

```
//    fluxo máximo
```

	x_{sv_1}	x_{sv_2}	$x_{v_1v_2}$	$x_{v_1v_3}$	$x_{v_1v_4}$	$x_{v_2v_4}$	$x_{v_3v_4}$	x_{v_3t}	x_{v_4t}	f		
vértice s	1	1								-1	=	0
vértice v_1	-1		1	1	1						=	0
vértice v_2		-1	-1			1					=	0
vértice v_3				-1			1	1			=	0
vértice v_4					-1	-1	-1		1		=	0
vértice t								-1	-1	1	=	0
arco x_{sv_1}	1										≤	2
arco x_{sv_2}		1									≤	3
arco $x_{v_1v_2}$			1								≤	2
arco $x_{v_1v_3}$				1							≤	3
arco $x_{v_1v_4}$					1						≤	1
arco $x_{v_2v_4}$						1					≤	2
arco $x_{v_3v_4}$							1				≤	4
arco x_{v_3t}								1			≤	1
arco x_{v_4t}									1		≤	3

Tabela 3.2: Restrições do problema de maximização de fluxo

```
/* função objectivo */
max: f;

/* Restrições */
//  conservação de fluxo
vertice_s: xs1 + xs2 = f;
vertice_1: xs1 = x12 + x13 + x14;
vertice_2: xs2 + x12 = x24;
vertice_3: x13 = x34 + x3t;
vertice_4: x14 + x24 + x34 = x4t;
vertice_t: x3t + x4t = f;

//  capacidade dos arcos
arco_s1: xs1 <= 2;
arco_s2: xs2 <= 3;
arco_12: x12 <= 2;
arco_13: x13 <= 3;
arco_14: x14 <= 1;
arco_24: x24 <= 2;
arco_34: x34 <= 4;
arco_3t: x3t <= 1;
arco_4t: x4t <= 3;
```

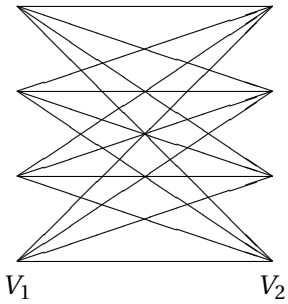
A solução óptima do problema tem um fluxo máximo $f^* = 4$. É fácil verificar que os arcos (s, v_2) e (v_2, v_4) não permitem que a rede trasporte mais fluxo. Teoria adicional sobre a relaão entre o fluxo máximo e o corte mínimo, apresentada na Secção 3.12, permite também verificar este resultado. □

3.5 Problema de afectação (*assignment*)

O objectivo do problema de afectação é afectar um determinado número de indivíduos n a um igual número de tarefas, de modo a minimizar os custos globais. Os custos associados à afectação do indivíduo i à tarefa j são conhecidos. O problema de afectação é um problema combinatório. Quando todos os indivíduos podem desempenhar qualquer uma das tarefas, existem $n!$ soluções possíveis. A sua solução por enumeração completa torna–se inviável mesmo para instâncias de dimensão reduzida.

O problema de afectação pode ser considerado como um caso particular do problema de transportes, em que as disponibilidades e as procuras são iguais à unidade.

O problema é definido num grafo bipartido, $G = (V_1, V_2, A)$, em dois conjuntos de vértices V_1 e V_2, representando cada vértice de V_1 um indivíduo e cada vértice de V_2 uma tarefa. Existe um custo de atribuir um indivíduo i a uma tarefa j, dado por c_{ij}. O esquema de afectação pode ser representado num grafo do tipo do apresentado na figura.



O problema de afectação pode ser formulado do seguinte modo:

$$\begin{aligned} \min \quad & \sum_{i \in V_1} \sum_{j \in V_2} c_{ij} x_{ij} \\ \text{suj. a} \quad & \sum_{j \in V_2} x_{ij} = 1 \ , \ \forall i \in V_1 \\ & \sum_{i \in V_1} x_{ij} = 1 \ , \ \forall j \in V_2 \\ & x_{ij} \geq 0 \end{aligned}$$

Exemplo 3.4. Considere um conjunto de tarefas que devem ser realizadas por um conjunto de indivíduos. A medida de tempos de desempenho de cada indivíduo na realização de cada tarefa é diferente, dependendo de aptidões específicas. Este problema pode ser também representado por um grafo bipartido, em que os custos associados aos arcos correspondem aos tempos que cada indivíduo necessita para desempenhar cada tarefa. O objectivo do problema consiste em determinar como se deveriam distribuir as pessoas pelas diversas tarefas de modo a minimizar os tempos globais de execução.

	A	B	C	D	
a	**2**	2	5	3	1
b	2	5	**4**	1	1
c	5	5	7	**2**	1
d	4	**3**	6	1	1
	1	1	1	1	

A solução óptima deste exemplo tem um custo total de 11, correspondendo à seguinte afectação: $(a, A), (b, C), (c, D)$ e (d, B).

□

Notas

No entanto, os algoritmos de transporte não se revelam eficientes para a sua solução. No problema de afectação existem apenas n variáveis positivas, iguais a 1, número bastante inferior ao valor normal de $n + n - 1$ variáveis básicas. Este facto torna as soluções do problema de transporte altamente degeneradas, sendo a maior parte das iterações consumidas em *pivôs* degenerados, sem haver melhoria da função objectivo.

O problema de afectação é um problema polinomial. O algoritmo húngaro, um algoritmo primal–dual, é polinomial, e resolve o problema de afectação reduzindo-o a uma sequência de problemas de emparelhamento.

3.6 Problema de gestão de pessoal

Há problemas de planeamento de pessoal em serviços de funcionamento diário permanente (*e.g.*, hospitais) que podem ser representados como um problema em rede. Isso ocorre quando a coluna que representa os períodos de serviço tem uma estrutura especial, com blocos de 1's consecutivos. É também permitido haver blocos que são partidos a meio à meia-noite (entre a última e a primeira linha da matriz).

Para ilustrar a estrutura em rede, vamos usar o exemplo anteriormente apresentado, adicionando variáveis de folga, e tornando as restrições do tipo ≥ em restrições de igualdade.

1	1	1			1				-1					*		=	6
2	1	1	1		1	1		1		-1					x		10
3	1	1	1	1		1	1				-1						14
4	1		1	1			1					-1			y		9
5	1			1									-1				8
6																	0

Subtraindo a cada linha a linha que lhe fica por cima, obtém-se:

1	1	1			1				-1					*		=	6
2			1			1		1	1	-1					x		4
3				1	-1		1	-1		1	-1						4
4		-1				-1					1	-1			y		-5
5			-1				-1					1	-1				-1
6	-1			-1									1				-8

A matriz tem agora colunas apenas com dois elementos diferentes de 0, sendo um deles igual a $+1$ e o outro a -1. Se associarmos cada coluna a um arco, o modelo pode ser representado numa rede, em que os valores associados aos vértices representam ofertas, quando positivos, ou procuras, quando negativos, como se mostra na Figura 3.3.

Se houver mais de um bloco de 1's consecutivos (*e.g.*, caso de haver intervalo para almoço, com interrupção de serviço), o modelo já não tem estrutura em rede.

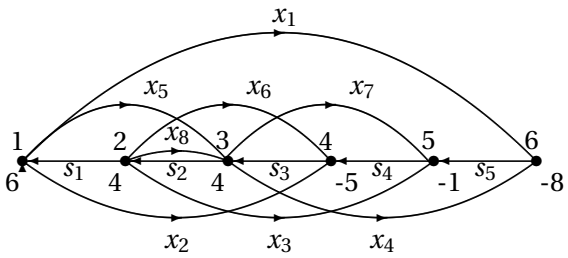


Figura 3.3: Rede do problema de gestão de pessoal com 1's consecutivos

3.7 Problema do saco de mochila: formulação II

O problema de saco de mochila pode ser formulado como um problema de fluxo em rede. Num problema de saco de mochila existe uma capacidade associada à mochila e itens com lucros e pesos associados. Dado que o modelo vai ser usado num modelo de corte e empacotamento, iremos usar a largura do rolo como capacidade da mochila e a largura como o peso dos itens, como se explica de seguida. Do mesmo modo, os itens colocados no saco de mochila constituirão um padrão de corte.

Dado um rolo de capacidade inteira W e um conjunto de diferentes larguras de pedidos $w_1, \ldots, w_d, \ldots, w_m$, o problema de determinar uma solução válida para um único padrão de corte pode ser modelado como um problema de encontrar um caminho num grafo acíclico orientado com $W+1$ vértices.

O modelo é representado num grafo $G = (V, A)$ com $V = \{0, 1, 2, \ldots, W\}$ e $A = \{(i, j) : 0 \leq i < j \leq W$ e $j - i = w_d$ para todo $d \leq m\}$, significando que existe um arco orientado entre dois vértices se existir um pedido da largura correspondente. O número de variáveis é $O(mW)$.

Há arcos adicionais entre $(k, k+1), k = 0, 1, \ldots, W-1$ correspondendo a porções não ocupadas do rolo. Existe um padrão de corte para um rolo se e só se existir um caminho entre os vértices 0 e W. Os comprimentos dos arcos que constituem o caminho definem as larguras dos itens que fazem parte do padrão de corte.

Exemplo 3.5. A Figura 3.4 mostra o grafo associado com uma instância com rolos de capacidade $W = 5$ e pedidos de largura 3 e 2. Na mesma Figura, é apresentado um caminho que corresponde a 2 pedidos de largura 2 e a 1 unidade de perda.

□

3.8 Problemas de corte e empacotamento

A formulação do problema de saco de mochila como um problema de determinar o caminho mais longo num grafo orientado pode ser usada para modelar problemas de corte e empacotamento. Se uma solução para um único rolo corresponde ao fluxo de uma unidade entre os vértices 0 e W, um caminho com um fluxo maior corresponderá a usar o mesmo padrão de corte em vários rolos. O mesmo acontece quando o fluxo representa vários padrões de corte.

A ideia principal é a seguinte: procura-se um fluxo numa rede em que a soma dos fluxos nos arcos de cada largura seja maior ou igual à respectiva quantidade pedida. Dada uma solução óptima inteira, pelo teorema de decomposição de fluxos, sabemos que existe uma decomposição inteira desse fluxo em caminhos, cada um deles correspondendo a um padrão de corte. De facto, pela

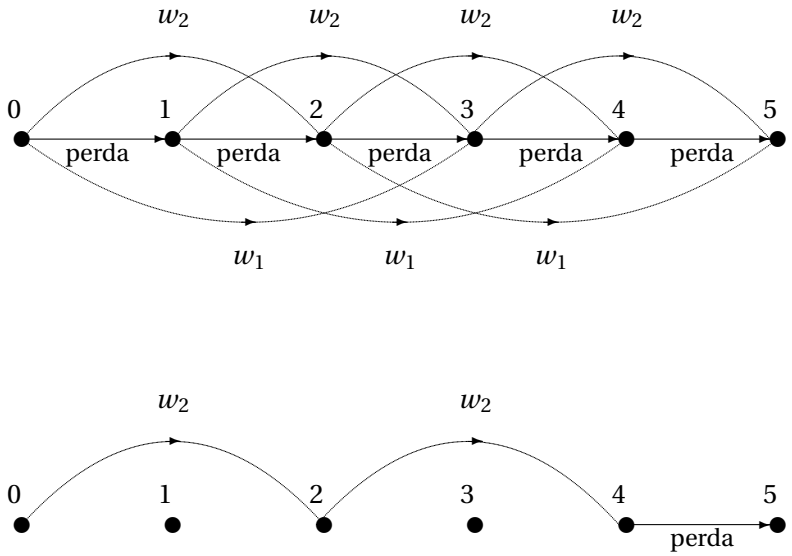


Figura 3.4: Grafo e padrão de corte

propriedade de decomposição de fluxo, fluxos não-negativos podem ser representados por caminhos e ciclos. O grafo G é acíclico, e, portanto, qualquer fluxo pode ser decomposto em caminhos orientados ligando o único nodo de oferta (nodo 0) ao único nodo de consumo (nodo W).

Este problema não pode ser resolvido usando um package de optimização de redes, porque as restrições adicionais destroem a estrutura em rede.

3.8.1 Formulação III - fluxos em arco com restrições adicionais

O problema é formulado como o problema de determinar o fluxo mínimo entre o vértice 0 e o vértice W com restrições adicionais que forçam que a soma de fluxos nos arcos de cada largura seja maior ou igual à respectiva quantidade pedida. Vamos considerar variáveis de decisão x_{ij}, associadas com os arcos definidos acima, que correspondem ao número de pedidos de largura $j-i$ colocados num qualquer rolo à distância de i unidades do princípio do rolo. A variável z pode ser vista como um arco de retorno, do vértice W para o vértice 0, e poderia ser também designada por x_{W0}. O modelo é o seguinte:

$$\min \quad z \tag{3.6}$$

$$\text{suj. a} \quad + \sum_{(i,j)\in A} x_{ij} - \sum_{(j,k)\in A} x_{jk} = \begin{cases} -z & \text{, if } j = 0 \\ 0 & \text{, if } j = 1,2,\ldots,W-1 \\ z & \text{, if } j = W \end{cases} \tag{3.7}$$

$$\sum_{(k,k+w_d)\in A} x_{k,k+w_d} \geq b_d \ , \ d = 1,2,\ldots,m \tag{3.8}$$

$$x_{ij} \geq 0 \ , \ \forall (i,j) \in A \tag{3.9}$$

$$x_{ij} \text{ inteiro} \ , \ \forall (i,j) \in A \tag{3.10}$$

Uma solução com valores inteiros de fluxo em todos os arcos pode ser transformada, usando a propriedade de decomposição de fluxo acima referida, numa solução inteira para o problema de corte.

Notas

3.9 Problema de fluxo multicomodidade

No modelo geral de problemas de fluxo em rede, salientou-se o facto de haver um único tipo de entidades. Há, no entanto, situações em que uma mesma rede suporta o fluxo de entidades de tipo diferente, ou há necessidade de diferenciar os fluxos. Isso ocorre, por exemplo, em problemas de comunicação, em que existe tráfego entre vários pares origem-destino, cada um desses pares designados por uma *stream* de tráfego, e é necessário diferenciar a que *stream* de tráfego um dado fluxo pertence. Para isso, usa-se um índice adicional na variável de decisão que identifica a *stream*.

Problemas deste tipo são designados na literatura anglo-saxónica por *multicommodity flow problem*. Na língua inglesa, *commodity* designa um bem ou uma mercadoria, mas a designação do problema em português usando uma palavra semelhante generalizou-se.

Pelo facto de normalmente terem associadas restrições adicionais, estes problemas deixam de poder ser resolvidos com *software* de optimização de redes, por deixarem de ser problemas de fluxo em rede puros.

Há várias formulações para este problema baseadas em fluxos em arco e em caminhos com variantes, por exemplo, consoante o fluxo de uma dada *stream* deve necessariamente usar um único caminhho na rede ou pode ser repartido por vários caminhos diferentes.

Iremos apresentar uma formulação baseada em fluxos em arco em que o fluxo pode ser repartido por vários caminhos.

3.9.1 Formulação - fluxos em arco com restrições adicionais

No problema de fluxo multicomodidades, há um conjunto K de comodidades, com elementos indexados por k, que fluem no grafo $G = (V, A)$. Cada comodidade k tem um fluxo positivo de q^k unidades desde o único vértice fonte dessa comodidade para o único vértice terminal da comodidade. O fluxo pode ser repartido por vários caminhos.

O vértice i tem um valor de:

$$b_i^k = \begin{cases} q^k & \text{, se o vértice } i \text{ for a fonte da comodidade } k \\ -q^k & \text{, se o vértice } i \text{ for o terminal da comodidade } k \\ 0 & \text{, caso contrário} \end{cases}$$

Comodidades que fluem num dado arco (i, j) partilham a sua capacidade, designada por u_{ij}. O problema pode ser formulado usando variáveis de decisão x_{ij}^k que designam o fluxo da comodidade k no arco (i, j). O modelo de programação linear é o seguinte:

$$\min \quad \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k \tag{3.11}$$

$$\text{suj. a} \quad + \sum_{(i,j) \in A} x_{ij}^k - \sum_{(j,i) \in A} x_{ji}^k = b_i^k, \ \forall i \in V, \forall k \in K \tag{3.12}$$

$$\sum_{k \in K} x_{ij}^k \leq u_{ij}, \ \forall (i,j) \in A \tag{3.13}$$

$$x_{ij}^k \geq 0, \ \forall k, \forall (i,j) \in A \tag{3.14}$$

As restrições (3.12) são *restrições de conservação de fluxo* que forçam, para cada comodidade k, que o fluxo que entra num vértice seja igual ao fluxo que sai (ofertas e procuras incluídas). Há

grupos destas restrições para cada comodidade e para vértice. As restrições (3.13) são *restrições de capacidade* que não permitem que o fluxo das várias comodidade que partilham um dado arco exceda a sua capacidade.

Para um exemplo de aplicação, ver a Secção 5.17.

3.10 Problema do caixeiro viajante

O Problema do Caixeiro Viajante tem um enunciado muito simples: dados um conjunto de cidades e as distâncias entre as cidades, determinar qual é o circuito de menor comprimento que passa por todas as cidades. A simplicidade de formulação contrasta com a dificuldade na sua resolução. Trata-se de um problema de programação inteira para o qual ainda não foi descoberto nenhum algoritmo eficiente de resolução, e crê–se que não exista (ver a Introdução do Capítulo 4).

Exemplo 3.6. Um exemplo de aplicação é a furação de placas de circuito impresso (PCB) em que se pretende efectuar um conjunto de n furos numa série de placas, utilizando uma máquina de controlo numérico. A máquina deve efectuar a sequência de furos numa placa, e voltar à posição inicial para voltar a desempenhar a mesma tarefa na placa seguinte. Os vértices correspondem às cidades a visitar e os custos correspondem às distâncias a percorrer, medidas no plano, para deslocar a máquina de furação entre quaisquer dois furos. □

Exemplo 3.7. Uma rede em anel estabelece ligações entre vários computadores ou servidores, de acordo com um protocolo bem definido. Neste tipo de rede, cada posto deve ser ligado exactamente a dois postos adjacentes. Se os custos de ligação entre cada par de postos forem dados, o problema pode ser resolvido como um problema de caixeiro viajante.

A opção de construção de rede em anel resulta da necessidade de evitar disrupções de serviço, assegurando a conectividade entre os servidores em caso de acidente, por exemplo, resultante do corte de um cabo. Houve acidentes deste tipo na área de comunicações que geraram perdas avultadas. Se a restrição de ligação a exactamente dois postos adjacentes fosse relaxada, a forma mais económica de interligar todos os postos seria construir a árvore de suporte de custo mínimo. □

Há várias formulações para o problema do caixeiro viajante. Apresenta-se uma formulação baseada no Problema de Afectação com restrições adicionais para eliminar subcircuitos. Este problema não pode ser resolvido usando um package de optimização de redes, porque as restrições adicionais destroem a estrutura em rede.

3.10.1 Circuitos Hamiltonianos

O Problema do Caixeiro Viajante consiste na determinação do Circuito Hamiltoniano de menor custo. Este nome deriva do matemático Sir William Hamilton, que inventou um jogo que consistia em fazer passar um fio pelos vértices de um dodecaedro. O dodecaedro possui circuitos Hamiltonianos. Um desses circuitos é obtido percorrendo os vértices segundo a ordem indicada na Figura 3.5.

O problema pode ser representado num grafo, orientado ou não, com custos associados às arestas. É usual dividir estes problemas em duas grandes classes, os problemas simétricos e assimétricos. No caso de problemas assimétricos, considera-se que as distâncias entre duas cidades podem ser diferentes, consoante os trajectos são percorridos num ou noutro sentido.

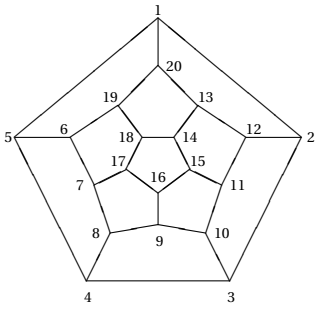


Figura 3.5: Dodecaedro

3.10.2 Formulação: solução do problema de afectação com restrições adicionais

O problema do caixeiro viajante tem uma estrutura muito semelhante à do problema de afectação. Muitas soluções do problema de afectação são também soluções válidas para o problema do caixeiro viajante. Há, no entanto, soluções do problema de afectação que não são soluções válidas para o problema do caixeiro viajante, porque não correspondem a um único circuito Hamiltoniano, mas sim a vários subcircuitos.

A formulação que vamos analisar, além das restrições do problema de afectação, tem também um conjunto de restrições para eliminar esses subcircuitos. Deste modo, são eliminadas as soluções do problema de afectação que não são válidas para o problema do caixeiro viajante.

Esta formulação pode ser utilizada, quer para o caso assimétrico, quer para o simétrico. No último caso, a matriz que define o problema deve ser uma matriz simétrica.

Considere um grafo orientado, $G = (V, A)$, constituído por n vértices e arcos orientados (i, j), a que estão associados custos c_{ij}. Seja x_{ij} uma variável binária que toma o valor 1 se o arco (i, j) é percorrido, *i.e.*, o caixeiro faz o percurso entre a cidade i e a cidade j, e o valor 0, caso contrário. Para forçar as variáveis x_{ii} a terem o valor nulo, são atribuídos custos suficientemente elevados aos coeficientes c_{ii}.

As restrições correspondentes ao problema de afectação são:

$$\min \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{3.15}$$

$$\text{suj. a} \quad \sum_{j=1}^{n} x_{ij} = 1 \ , \ \forall i \tag{3.16}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \ , \ \forall j \tag{3.17}$$

$$x_{ij} \geq 0 \tag{3.18}$$

O grupo de restrições (3.16) significa que, de todos os arcos que saem do vértice i, apenas um deles pode pertencer ao circuito Hamiltoniano. Por outro lado, as restrições (3.17) apenas permitem que, dos arcos que entram no vértice j, um possa pertencer ao circuito.

Os problemas surgem porque há várias soluções válidas para o problema de afectação que não são válidas para o problema do caixeiro viajante. Essas soluções correspondem a um conjunto de vários subcircuitos, e não a um circuito único que visite todas as cidades.

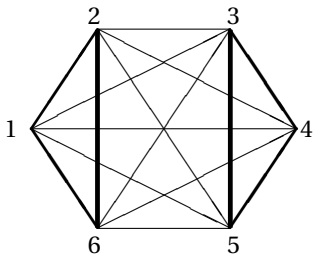


Figura 3.6: Solução do Problema de Afectação com Dois Subcircuitos

Exemplo 3.8. Considere o problema com 6 vértices apresentado na Figura 3.6. A solução $x_{12} = x_{26} = x_{61} = x_{34} = x_{45} = x_{53} = 1$ constitui uma solução válida para o problema de afectação, mas não é um percurso válido para o problema do caixeiro viajante.

□

Para obter a formulação completa do problema do caixeiro viajante, é necessário adicionar restrições que evitem a formação de subcircuitos. Estas restrições podem ser expressas de vários modos, havendo tantas restrições quantos os subconjuntos possíveis de vértices. Considere um subconjunto próprio S dos vértices de V, *i.e.*, um conjunto não vazio e diferente do próprio V. Para qualquer subconjunto de vértices S, o número de arcos deve ser menor que o número de vértices, *i.e.*,

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |\, S \,| - 1 \;,\; \forall S \subset V, S \neq \emptyset \tag{3.19}$$

Se existir um subcircuito, haverá um conjunto de s vértices entre os quais existem s arcos. O significado da Equação (3.19) pode ser interpretado do seguinte modo: das s variáveis, uma deve tomar o valor nulo.

Exemplo 3.9. Para $S = \{1,2,6\}$, o número de arcos na solução deve ser inferior a 2, *i.e.*,

$$x_{12} + x_{16} + x_{26} + x_{21} + x_{61} + x_{62} \leq 2.$$

□

Outra forma de exprimir as restrições de eliminação de subcircuitos consiste em afirmar que deve existir, pelo menos, um arco em cada corte, *i.e.*, entre S e $\bar{S}$, o complemento de S:

$$\sum_{i \in S} \sum_{j \in \bar{S}} x_{ij} \geq 1 \;,\; \forall S \subset V, S \neq \emptyset$$

Exemplo 3.10. Para $S = \{1,2,6\}$, sendo $\bar{S} = \{3,4,5\}$, deve existir, pelo menos, um arco entre estes dois conjuntos, *i.e.*,

$$x_{13} + x_{14} + x_{15} + x_{23} + x_{24} + x_{25} + x_{63} + x_{64} + x_{65} \geq 1.$$

□

A consideração de qualquer um destes conjuntos de restrições para todos os subconjuntos próprios de V, juntamente com as restrições do poliedro de afectação, daria o poliedro do problema do caixeiro viajante. Em qualquer dos casos, o número de restrições é exponencial, relativamente ao número de vértices. Sendo $n = \mid V \mid$, sabe-se que há, no total, 2^n subconjuntos de V. Não sendo considerados o conjunto vazio, nem o próprio V, há $2^n - 2$ restrições de eliminação de subcircuitos, cada uma correspondendo a um subconjunto próprio S diferente. A solução do problema baseada numa formulação que incluísse todas estas restrições não é viável do ponto de vista prático.

Notas

Nas Secções 5.11, 5.12 e 5.12 apresentam-se casos de aplicação de Recolha de materiais num armazém, Sequenciação de tarefas com custos dependentes da sequência e Sequenciação em sistemas *Flowshop* sem Inventários Intermédios, respectivamente.

O Problema do Caixeiro Viajante constitui em certos algoritmos, um subproblema de problemas de distribuição e planeamento de rotas de veículos. O problema de planeamento de rotas consiste em determinar, para um dado conjunto de veículos, qual o percurso que cada veículo deve efectuar, de modo a, no seu conjunto, servir todos os clientes. Estes problemas podem, ainda, ter restrições suplementares, como limitações à capacidade dos veículos ou janelas temporais, que determinam o período admitido para a visita a determinadas cidades.

3.11 Afastamento máximo de dois vértices: dual do caminho mais curto

O objectivo deste problema é maximizar o afastamento de dois vértices duma rede, designados por s e t, respeitando a restrição de que dois quaisquer vértices do grafo não podem estar mais afastados entre si do que o comprimento da aresta que os une. Existe um modelo analógico para este problema, que envolve a construção de uma rede, semelhante a uma rede de pesca, em que os vértices são unidos por fios com um comprimento igual ao custo da aresta. A solução óptima do problema pode ser determinada com o modelo analógico segurando os vértices s e t com mãos diferentes, e afastando as mãos. Os fios que ficam esticados formam o caminho mais curto, que tem um comprimento igual ao afastamento máximo dos vértices. Um facto interessante é que o afastamento máximo de dois vértices é a solução óptima do problema dual do problema do caminho mais curto, como se detalha em baixo. O modelo é o seguinte:

$$\begin{array}{ll} \max & d_t - d_s \\ \text{suj. a} & -d_i + d_j \leq c_{ij}\ ,\ \forall (i,j) \in A\ ,\ i,j \in V \\ & d_j \text{ não restringido em sinal }\ ,\ \forall\, j \in V \end{array}$$

designando d_i a distância do vértice i ao vértice de origem s.

O problema tem uma solução óptima, mas há uma infinidade de soluções óptimas alternativas consoante o valor da variável de decisão d_s. Por isso, podemos fixar o valor $d_s = 0$, ficando o objectivo do problema reduzido a maximizar d_t. As restrições do problema são $d_j \leq d_i + c_{ij}$. Cada restrição significa que, se existir um arco entre os vértices i e j, o vértice j não pode ter um afastamento maior do vértice i do que o do comprimento do arco que os une. O objectivo é maximizar d_t, ou seja, determinar o afastamento máximo que o vértice t pode ter em relação ao vértice s.

Exemplo 3.11. Considere o exemplo apresentado no problema de caminho mais curto.

```
/* função objectivo */
max: dt ;

/* Restrições */
posi_vs: ds = 0;
arco_s0: d0 <= ds + 0 ;
arco_s6: d6 <= ds + 0 ;
arco_01: d1 <= d0 + 4 ;
arco_04: d4 <= d0 + 4 ;
arco_12: d2 <= d1 + 6 ;
arco_23: d3 <= d2 + 7 ;
arco_3t: dt <= d3 + 2 ;
arco_42: d2 <= d4 + 9 ;
arco_45: d5 <= d4 + 9 ;
arco_53: d3 <= d5 + 4 ;
arco_5t: dt <= d5 + 4 ;
arco_610: d10 <= d6 + 5 ;
arco_67: d7 <= d6 + 5 ;
arco_74: d4 <= d7 + 6 ;
arco_78: d8 <= d7 + 6 ;
arco_85: d5 <= d8 + 4 ;
arco_89: d9 <= d8 + 4 ;
arco_9t: dt <= d9 + 2 ;
arco_1011: d11 <= d10 + 8;
arco_108: d8 <= d10 + 8 ;
arco_119: d9 <= d11 + 7 ;
```

A solução óptima do problema dada pelo LPSolve é:

Variables	result
	17
dt	17
d3	15
d9	15
d5	13
d8	11
d2	8
d11	8
d7	5
d4	4
d10	3
d1	2
ds	0
d0	0
d6	0

Notas

O modelo apresentado pode ser obtido construindo o modelo dual do problema do caminho mais curto, depois de multiplicar todas as restrições de igualdade do problema primal por -1. Cada

restrição do problema primal é relativa à conservação de fluxo num vértice j, que terá uma variável dual associada designada por d_j.

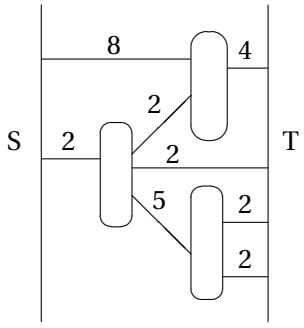
O próprio facto de se fixar a variável $d_s = 0$ tem uma interpretação. As restrições do problema primal não são linearmente independentes, porque a soma de todas as linhas é igual a 0. Sendo uma das restrições linearmente dependente, a sua remoção corresponde a fixar o valor da variável dual que lhe está associada em 0.

Do ponto de vista do problema do caminho mais curto, as restrições $d_j \leq d_i + c_{ij}$ representam condições de optimalidade. Claramente, se existir algum arco (i, j) em que esta condição não seja cumprida, a solução não é óptima, porque podemos associar um outro valor ao vértice j correspondendo a uma distância mais curta para esse vértice.

3.12 Problema do corte mínimo: dual do fluxo máximo

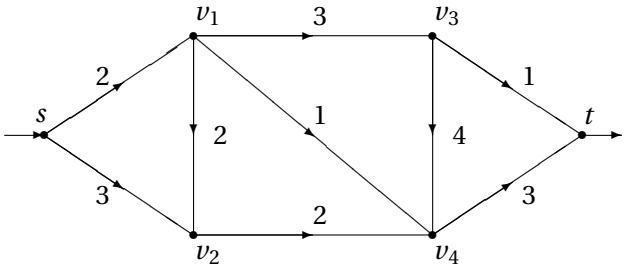
O problema do corte mínimo de um grafo consiste em determinar o conjunto de arcos com um custo total mínimo que, ao serem removidos, isolam a fonte do terminal do grafo, não permitindo a circulação de fluxo.

Exemplo 3.12. Durante a batalha do rio Quei, as tropas do General S avançaram até à margem oeste do rio, cuja carta é apresentada. Nesta zona do rio, a única em que é possível efectuar uma travessia num raio de muitas centenas de kilómetros, há várias ilhas, e pontes entre as margens e as ilhas. O General T, nosso firme aliado, pretende isolar as duas margens, dinamitando algumas pontes. Ele estima que os Kg de TNT necessários para tornar cada ponte inoperacional são os indicados na Figura.



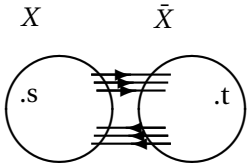
Determinar o corte mínimo permite ao General T identificar as pontes que devem ser dinamitadas para travar o avanço do inimigo e saber quantos kilos de TNT deve requisitar ao Quartel General para esta operação. □

Exemplo 3.13. Para a rede apresentada na Figura 3.2, que aqui se reproduz, a remoção simultânea dos arcos (v_1, v_3), (v_1, v_4) e (v_2, v_4), isola a fonte s do terminal t, embora esta solução, com um custo igual a 6, não seja a que minimiza os custos.



□

Definição 1. Um corte $(X, \bar{X})$ é uma partição do conjunto de vértices V em dois conjuntos disjuntos X e $\bar{X}$, devendo $s \in X$ e $t \in \bar{X}$.



Definição 2. O fluxo num corte $(X, \bar{X})$, designado por $f(X, \bar{X})$, é igual à quantidade líquida de fluxo que atravessa o corte no sentido de X para $\bar{X}$.

$$f(X, \bar{X}) = \sum_{i \in X, j \in \bar{X}} x_{ij} - \sum_{i \in \bar{X}, j \in X} x_{ij}$$

Definição 3. A capacidade do corte $(X, \bar{X})$ é igual à soma das capacidades dos arcos cuja origem é um vértice de X e cujo destino é um vértice de $\bar{X}$, *i.e.*,

$$cap(X, \bar{X}) = \sum_{i \in X, j \in \bar{X}} l_{ij}$$

Exemplo 3.14. Para o Exemplo introduzido,

$$X = \{s, v_1, v_2\}, \bar{X} = \{v_3, v_4, t\}, cap(X, \bar{X}) = 6.$$

$$X = \{s, v_2\}, \bar{X} = \{v_1, v_3, v_4, t\}, cap(X, \bar{X}) = 4.$$

□

Lema 1. O valor de qualquer fluxo válido, $f(X, \bar{X})$, é menor ou igual que a capacidade de qualquer corte, $cap(X, \bar{X})$:

$$f(X, \bar{X}) \leq cap(X, \bar{X})$$

Prova: Seja $(X, \bar{X})$ um qualquer corte. Se somarmos as equações de conservação de fluxo relativas aos vértices pertencentes ao conjunto X, obtém-se a seguinte Equação:

$$\sum_{i \in X, j \in \bar{X}} x_{ij} - \sum_{i \in \bar{X}, j \in X} x_{ij} = f(X, \bar{X})$$

Como $0 \leq x_{ij} \leq l_{ij}$, a quantidade de fluxo que sai do conjunto X é menor que a soma das capacidades dos arcos que saem de X. Também, o fluxo nos arcos de retorno, provenientes de $\bar{X}$, é maior do que 0. Agregando as condições relativas aos arcos que atravessam o corte, obtém-se:

$$\sum_{i \in X, j \in \bar{X}} x_{ij} \leq \sum_{i \in X, j \in \bar{X}} l_{ij} \quad \text{e} \quad \sum_{i \in \bar{X}, j \in X} x_{ij} \geq 0$$

Conjugando as condições anteriores, é válida a seguinte relação:

$$\begin{aligned} \sum_{i\in X, j\in \bar{X}} l_{ij} - 0 &\geq f(X,\bar{X}) \\ cap(X,\bar{X}) &\geq f(X,\bar{X}) \end{aligned}$$

□

Se todos os arcos que atravessam o corte no sentido de X para $\bar{X}$ estiverem saturados e o fluxo nos arcos que atravessam o corte em sentido contrário for nulo, a inequação toma a forma de uma igualdade. Este teorema é conhecido na literatura anglo–saxónica com o *maxflow–mincut theorem*.

Teorema 2. Em qualquer rede, o valor do fluxo máximo é igual à capacidade do corte mínimo.

Formulação do dual

Associando variáveis duais u às restrições de conservação de fluxo nos vértices e variáveis duais v às restrições de capacidade dos arcos, a formulação dual do problema de fluxo máximo é:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} l_{ij} v_{ij} \\ \text{suj. a} \quad & u_t - u_s = 1 \\ & u_i - u_j + v_{ij} \geq 0\,,\ \forall (i,j)\in A\,,\ i,j\in V \\ & v_{ij} \geq 0\,,\ \forall (i,j)\in A \end{aligned}$$

As variáveis duais correspondentes às restrições de conservação de fluxo são não–restringidas, enquanto as variáveis duais correspondentes às restrições de capacidade são não–negativas.

Existe uma equivalência entre cortes $(X,\bar{X})$ e soluções válidas do problema dual. O valor da função objectivo da solução válida é igual à capacidade do corte $(X,\bar{X})$. Dado um qualquer corte, é possível construir uma solução válida para o problema dual do seguinte modo:

$$\begin{aligned} u_i &= \begin{cases} 0 & \text{, se } i\in X \\ 1 & \text{, se } i\in \bar{X} \end{cases} \\ v_{ij} &= \begin{cases} 1 & \text{, se } (i,j): i\in X, j\in \bar{X} \\ 0 & \text{, caso contrário} \end{cases} \end{aligned}$$

Exemplo 3.15. Para a rede apresentada na Figura 3.2, o modelo é o seguinte:

```
/* Corte mínimo          */

/* funcão objectivo    */
min:   2 vs1 + 3 vs2 + 2 v12 + 3 v13 + 1 v14
      +2 v24 + 4 v34 + 1 v3t + 3v4t;

/* restrições */
ts:        ut - us = 1;
arco_s1: us - u1 + vs1 >= 0 ;
arco_s2: us - u2 + vs2 >= 0 ;
```

```
arco_12: u1 - u2 + v12 >= 0 ;
arco_13: u1 - u3 + v13 >= 0 ;
arco_14: u1 - u4 + v14 >= 0 ;
arco_24: u2 - u4 + v24 >= 0 ;
arco_34: u3 - u4 + v34 >= 0 ;
arco_3t: u3 - ut + v3t >= 0 ;
arco_4t: u4 - ut + v4t >= 0 ;

free us, u1, u2, u3, u4, ut;
```

A solução óptima do problema de corte mínimo dada pelo LPSolve tem um valor igual a 4, sendo as variáveis de decisão $v_{s1} = v_{24} = u_1 = u_3 = u_4 = u_t = 1$, e as restantes variáveis iguais a 0. Esta solução corresponde ao corte mínimo $X = \{s, v_2\}$,$\bar{X} = \{v_1, v_3, v_4, t\}$. A capacidade do corte $(X, \bar{X})$ é igual a 4, o mesmo valor do fluxo máximo, o problema dual. □

Da teoria da dualidade, sabe-se que o valor de qualquer solução válida do problema primal de maximização é sempre menor ou igual ao valor de qualquer solução válida do problema dual. As soluções dos problemas primal e dual correspondem, respectivamente, a fluxos e a cortes, pelo que a teoria da dualidade fornece uma prova alternativa para o Teorema apresentado na Secção **??**.

3.13 Transformações básicas

Há problemas em que se pretende modelar situações reais que correspondem a ter uma capacidade associada a um vértice ou seja requerido que um dado arco tenha um valor de fluxo maior ou igual a um limite inferior.

As transformações apresentadas de seguida aplicadas sucessivamente a cada situação acima descrita permitem transformar o grafo, *i.e.*, criar um novo grafo em que todos os arcos apenas têm limites superiores, sendo os limites inferiores nulos, como é habitual em problemas de programação linear e como se usa no modelo geral de fluxos em rede.

Da mesma forma, o formato normalmente usado em *software* de optimização de redes (*e.g.*, *relax4*) define o grafo $G = (V, A)$ através de uma lista de arcos orientados, cada um deles identificado pelos quatro componentes $(i, j, c_{ij}, u_{ij}), \forall (i, j) \in A$, sendo i a origem do arco, j o destino do arco e c_{ij} o custo unitário de transporte no arco, e u_{ij} o limite superior de fluxo no arco.

3.13.1 Arestas

Há problemas que são definidos num grafo não-orientado, mas a situação real requer ou permite que a aresta possa ser percorrida nos dois sentidos ou que haja fluxo em qualquer sentido.

Nesse caso, a aresta (arco não orientado) deve ser substituída por um par de arcos, cada um deles um arco orientado, conforme se apresenta na Figura 3.7, em que c_{ij} é o custo unitário de transporte no arco.

3.13.2 Capacidades nos vértices

Há situações em que é necessário associar uma capacidade a um vértice de um grafo. Tal ocorre, por exemplo, quando o vértice representa um recurso que tem capacidade limitada e tem de processar entidades que passam pelo vértice. São exemplos disso situações em que o vértice repre-

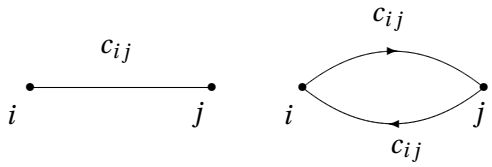


Figura 3.7: Transformação de arestas em arcos orientados

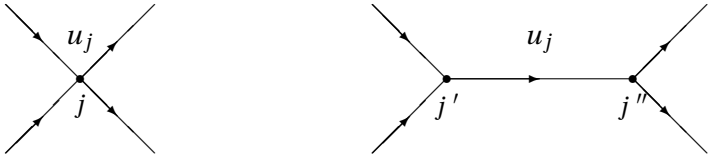


Figura 3.8: Transformação da capacidade de vértice na capacidade de um arco

senta um armazém intermédio de uma plataforma logística multimodal, onde se articulam diversos meios de transporte, ou um *router* de uma rede de comunicação, que é o dispositivo que limita a capacidade de transmissão, dado que há tecnologias para as linhas de comunicação que tornam a sua capacidade virtualmente infinita.

Os limites impostos ao número de unidades que podem passar através de um vértice podem ser transformados em limites superiores de arcos.

Dado um grafo $G = (V, A)$, uma capacidade C_j^{max} associada ao vértice j significa que o fluxo de entrada e o de saída não podem exceder a capacidade:

$$\sum_{(i,j)\in A} x_{ij} = \sum_{(j,k)\in A} x_{jk} \leq C_j^{max}.$$

A transformação consiste em desdobrar o vértice j em dois vértices j' e j'' e em criar um novo arco entre os dois novos vértices com um limite superior igual à capacidade do vértice, como se mostra na Figura 3.8. Todos os arcos cujo destino é j passam a ser incidentes em j' e todos os arcos cuja origem é j passam a ser incidentes em j''.

Exemplo 3.16. Um exemplo ilustrativo é apresentado na Figura 3.9. Os valores associados aos arcos (c_{ij}, u_{ij}), sendo c_{ij} o custo unitário de transporte, u_{ij} o limite superior de fluxo no arco e C_j^{max} o fluxo máximo no vértice j.

□

Exemplo 3.17. Existem armazéns a e b entre as origens e os destinos com capacidades de 10 e de 16, respectivamente. Cada vértice representando um armazém é desdobrado num vértice de entrada e num vértice de saída, e é criado um arco com a capacidade do armazém.

O fluxo pelo armazém é limitado pela sua capacidade. O custo do novo arco tipicamente é nulo; no entanto, pode ser igual ao custo unitário de armazenagem.

□

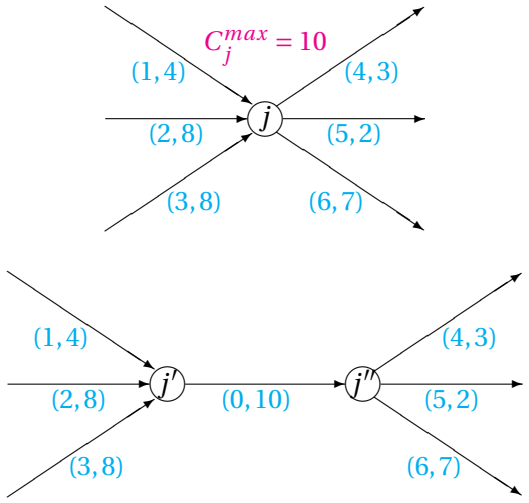


Figura 3.9: Exemplo: transformação da capacidade de vértice na capacidade de um arco

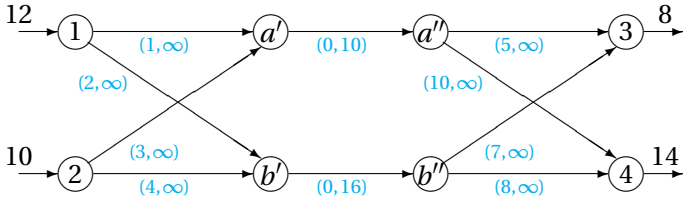


Figura 3.10: Exemplo: armazéns intermédios com capacidade

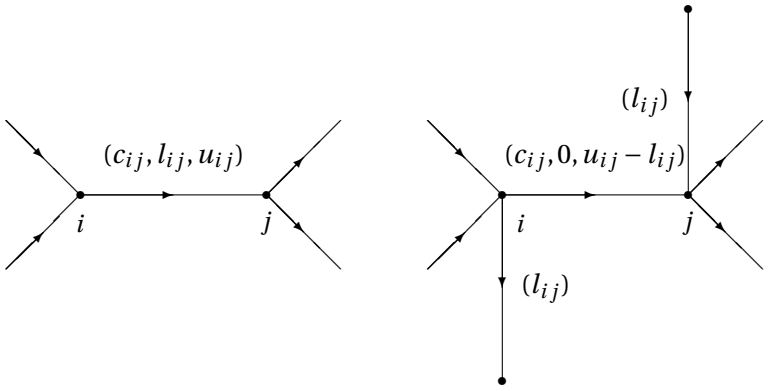


Figura 3.11: Arcos com limites inferiores

3.13.3 Limites inferiores nos arcos

É possível transformar uma instância com limites inferiores (l_{ij}) positivos numa instância com limites inferiores nulos em todos os arcos.

Dado um grafo $G = (V, A)$, pretende-se que haja um fluxo mínimo no arco $(i, j) : x_{ij} \geq l_{ij}$.

Exemplo 3.18. No exemplo apresentado na Figura3.12, os valores associados aos arcos (c_{ij}, l_{ij}, u_{ij}) representam o custo unitário de transporte, c_{ij}, o limite inferior de fluxo no arco, l_{ij}, e o limite superior de fluxo no arco, u_{ij}. Também se assume que não há oferta nem consumo nos vértices$i$ e j.

No exemplo considerámos que as ofertas e consumos no vértices i e j eram nulos. No caso geral, é necessário reajustar os valores das ofertas / procuras.

Os valores da oferta / procura nos vértices i e j devem ser reajustados. A procura do vértice i é aumentada de l_{ij} unidades, passando a ser igual a $b_i + l_{ij}$ unidades. Caso o vértice i seja um vértice em que há uma oferta de valor b_i, o novo valor da oferta passa a ser igual a $b_i - l_{ij}$ unidades. Se o resultado final da subtracção for um valor negativo, o vértice deixa de ser um vértice em que há oferta e passa a ser um vértice de consumo. O mesmo tipo de ajustamento dve ser feito no vértice j, sendo a oferta do vértice j é aumentada de l_{ij} unidades.

Esta transformação é equivalente a efectuar uma mudança de variável $x'_{ij} = x_{ij} - l_{ij}$ no modelo de programação linear do problema de fluxo em rede.

Claramente, após calcular a solução óptima do problema transformado, os valores finais do fluxo no arco devem ser recalculados, bem como os custos.

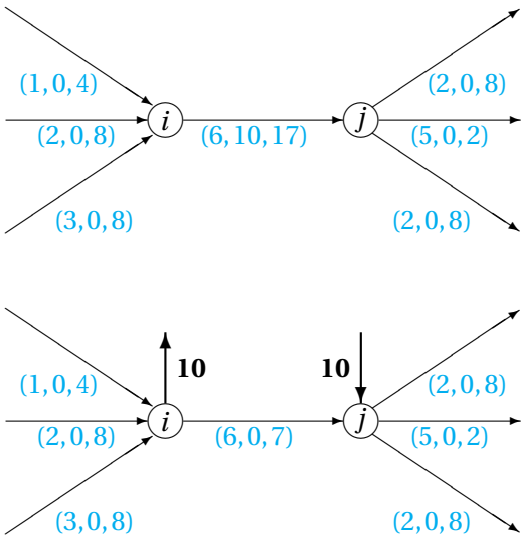


Figura 3.12: Exemplo: arcos com limites inferiores

Capítulo 4

Programação inteira com restrições lógicas

Há modelos em que as variáveis de decisão expressam, por exemplo, as quantidades a produzir de um artigo. Usando programação linear, a resolução destes modelos pode produzir uma solução óptima com variáveis fraccionárias. Desejavelmente, estes valores devem ser inteiros, mas frequentemente o arredondamento das variáveis fraccionárias produz soluções de muito boa qualidade.

A importância da programação inteira não advém desse tipo de problemas, mas sim da necessidade de modelar aspectos que não podem ser representados através de um conjunto de restrições lineares. Isso ocorre, por exemplo, quando se abordam problemas reais, e se pretende modelar uma função objectivo que não é linear ou explorar um domínio que não é convexo. Os modelos resultantes têm tipicamente um conjunto de variáveis binárias e um conjunto de outras variáveis, que, mesmo que na situação real devam ser inteiras, são declaradas no modelo como podendo ser fraccionárias. É que o seu arredondamento é aceitável. O mesmo não se passa com as variáveis binárias, porque se o seu valor for arredondando, muitas vezes as soluções resultantes não são admissíveis.

Neste capítulo, apresentam-se várias formulações típicas de programação inteira, a maioria das quais utiliza variáveis binárias para traduzir as não-linearidades surgidas no mundo real. O uso de variáveis binárias permite introduzir nos modelos relações do domínio da Lógica.

Deixa-se aqui uma nota em relação à dificuldade em resolver problemas de programação inteira. Os problemas de programação linear, como aqueles em que todas as variáveis podem tomar valores fraccionários ou os de fluxos em rede, são problemas para os quais são conhecidos algoritmos de resolução em tempo polinomial. Assim, é possível resolver instâncias de muito grande dimensão.

Por outro lado, o problema de programação inteira é um problema $NP-$completo, e existem boas razões para supor que os problemas que pertencem a esta classe não podem ser resolvidos em tempo polinomial usando computação determinística. A existência de um algoritmo polinomial para problemas desta classe é um problema que continua em aberto.

As técnicas usadas em Investigação Operacional (muitas das quais não são abordadas nesta unidade curricular, que tem carácter introdutório) estão entre as de maior sucesso na resolução de problemas de programação inteira. Um dos aspectos que é crucial para esse sucesso é a qualidade da formulação, que, de uma forma genérica, está relacionada com a diferença dos valores da solução óptima inteira e da solução da relaxação linear do problema inteiro.

Este tópico está fora do âmbito desta unidade curricular, mas pode ser ilustrado pelos exemplos das diferentes formulações para o problema de corte e empacotamento. É possível resolver instâncias de muito maior dimensão usando as formulações de fluxo em arco ou de padrões de corte do que usando a formulação com dois índices.

4.1 Modelo geral

O problema geral de programação linear inteira mista pode ser formulado do seguinte modo:

$$\begin{aligned} \max z_I &= cx + dy \\ \text{suj. a} \quad & Ax + Dy = b \\ & x \geq 0 \text{ e inteiro}, y \geq 0 \end{aligned}$$

sendo $A \in \mathbb{R}^{m \times n_1}, D \in \mathbb{R}^{m \times n_2}, c \in \mathbb{R}^{1 \times n_1}, d \in \mathbb{R}^{1 \times n_2}, x \in \mathbb{Z}_+^{n_1 \times 1}, y \in \mathbb{R}_+^{n_2 \times 1}, b \in \mathbb{R}^{m \times 1}$.

4.2 Expressões lógicas e restrições com variáveis binárias

As expressões lógicas podem ser expressas como restrições com variáveis binárias. Os valores 1 e 0 das variáveis binárias correspondem aos valores lógicos V e F, respectivamente. A expressão lógica é verdadeira se e só se a correspondente restrição com variáveis binárias for obedecida.

Considere um conjunto de literais $\{x_1, x_2, \ldots, x_n\}$, cada um podendo tomar o valor lógico verdadeiro (V) ou falso (F) e sendo $\overline{x}_i$ o complemento (negação) do literal x_i. O símbolo $\wedge$ designa a operação lógica *e*. O símbolo $\vee$ designa a operação lógica *ou*. A expressão lógica toma o valor verdadeiro, V, quando o função linear envolvida na restrição toma um valor maior ou igual à unidade, cujo equivalente lógico é V.

Exemplo 4.1. Se pretendermos que uma expressão lógica seja verdadeira, podemos traduzi-la usando uma restrição com variáveis binárias cujo resultado seja verdadeiro.

Expressão lógica	Restrição binária
$a \vee b \vee c$	$a + b + c \geq 1$
$\overline{a} \vee b$	$(1 - a) + b \geq 1$

□

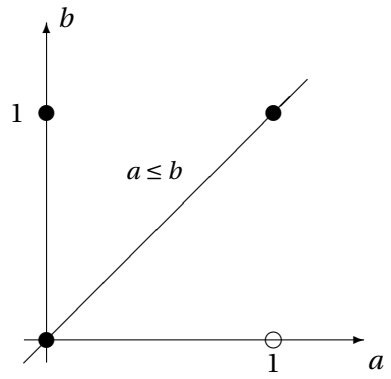
4.2.1 Restrições com relações entre variáveis binárias

Implicação lógica

Um dos tipos mais utilizado é a implicação lógica, para exprimir, por exemplo, que, se a actividade a é seleccionada, então a actividade b deve ser seleccionada. À implicação $a \Rightarrow b$, corresponde a restrição com variáveis binárias $a \leq b$, dado terem a mesma tabela lógica, como se pode verificar no seguinte quadro:

a	b	$a \Rightarrow b$	a	b	$a \leq b$
F	F	V	0	0	OK
F	V	V	0	1	OK
V	F	F	1	0	$-$
V	V	V	1	1	OK

A restrição que relaciona as variáveis binárias a e b é a seguinte:



Sabe-se que há várias formas equivalentes de traduzir uma implicação lógica, todas elas conduzindo à mesma restrição com variáveis binárias $a \leq b$.

Expressão lógica	Restrição binária
$a \Rightarrow b$	$a \leq b$
$\overline{a} \vee b$	$(1-a)+b \geq 1$
$\overline{b} \Rightarrow \overline{a}$	$(1-b) \leq (1-a)$

Outros exemplos

O uso de restrições com variáveis binárias permite estabelecer outras relações como:

Expressão lógica	Restrição binária
$a \vee b$	$a+b \geq 1$
$a \Rightarrow b$	$a \leq b$
$a \Rightarrow \overline{b}$	$a+b \leq 1$
$b \Rightarrow \overline{a}$	$a+b \leq 1$
$a \dot{\vee} b$ (ou exclusivo)	$a+b=1$
seleccionar *exactamente* uma das opções	$a+b+\ldots+z=1$
seleccionar, *no máximo.* uma das opções	$a+b+\ldots+z \leq 1$
$a.b \Rightarrow c$	$a+b-1 \leq c$

Exemplo 4.2. É possível modelar uma variável pode ter um conjunto discreto finito de valores„ como se pode mostra nos seguinte exemplos. Para a variável $x \in \{1,2,3,5\}$, usa-se:

$$\begin{aligned} x &= 1y_1 + 2y_2 + 3y_3 + 5y_4 \\ & y_1 + y_2 + y_3 + y_4 = 1 \\ & y_1, y_2, y_3, y_4 \in \{0,1\} \end{aligned}$$

Para a variável $x \in \{0,1,2,3,5\}$, usa-se:

$$\begin{aligned} x &= 1y_1 + 2y_2 + 3y_3 + 5y_4 \\ & y_1 + y_2 + y_3 + y_4 \leq 1 \\ & y_1, y_2, y_3, y_4 \in \{0,1\} \end{aligned}$$

□

4.2.2 Restrições com Grande M

Além de relações lógicas entre eventos, cada um deles representado por uma variável binária, há situações em que é necessário estabelecer uma relação entre uma variável inteira e uma variável binária que sinaliza um determinado evento.

Isso ocorre no seguinte exemplo, em que, quando a variável que indica o número de unidades produzidas assume um valor positivo, se força uma variável binária, que sinaliza que existe produção a ter o valor 1. Por outro lado, se a produção for nula, a variável binária pode assumir o valor 0. Há interesse em usar a variável binária de sinalização, porque ela pode ser usada, por exemplo, para adicionar à função objectivo um custo de preparação, necessário ao início da produção.

Seja x a variável que indica o número de unidades produzidas e y a variável binária que sinaliza que existe produção. A relação entre as duas variáveis é expressa, de uma forma genérica, do seguinte modo:

$$x \leq My$$

em que M deve ser um valor suficientemente elevado.

Esta restrição assegura que, sempre que $x > 0$, a variável de decisão binária $y = 1$. Quando $x = 0$, a variável binária y pode assumir o valor nulo, e isso acontece, por essa opção ser mais favorável quando a variável é associado a um custo.

Claramente, temos de assumir que a variável x não pode tomar um valor ilimitado, e isso tipicamente acontece em problemas reais. O valor de M deve ser suficientemente elevado para que nenhum valor possível para a variável x seja eliminado, por uma restrição que se destina somente a criar a sinalização de um evento. Se for conhecido um limite superior para a variável x, podemos seleccionar esse limite superior para o valor de M. Caso contrário, usa-se um valor que a variável x não possa previsivelmente atingir.

4.3 Problema da satisfação booleana

O problema da *Satisfação de uma expressão lógica* ou da *Satisfação booleana* é um *problema de decisão* da área da Lógica que visa determinar se existe alguma atribuição de valores lógicos aos literais que tornem a expressão lógica verdadeira.

Pode provar-se que qualquer expressão lógica pode ser expressa como a conjunção de um número finito de disjunções, *cláusulas lógicas*, onde cada literal aparece apenas uma vez. Diz-se que a expressão está na *Forma normal conjuntiva*. Para a expressão lógica ser satisfeita, todas as cláusulas lógicas devem ser satisfeitas.

Exemplo 4.3. Considere-se a seguinte expressão lógica.

$$(x_1 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_2) \wedge (x_2 \vee x_3)$$

Os valores lógicos x_2 =V e $x_1 = x_3$ =F satisfazem essa expressão. Do ponto de vista de programação inteira, a expressão é satisfeita se existir alguma solução admissível do seguinte problema:

$$\begin{cases} x_1 + (1 - x_3) \geq 1 \\ (1 - x_1) + x_2 \geq 1 \\ x_2 + x_3 \geq 1 \\ x_1, x_2, x_3 \text{ binárias} \end{cases} \qquad \begin{cases} x_1 - x_3 \geq 0 \\ -x_1 + x_2 \geq 0 \\ x_2 + x_3 \geq 1 \\ x_1, x_2, x_3 \text{ binárias} \end{cases}$$

O ponto $(x_1, x_2, x_3)^\top = (0, 1, 0)^\top$ é uma solução admissível do domínio definido pelas restrições. Pode usar-se qualquer função objectivo, porque queremos apenas encontrar uma solução admissível. □

O problema da satisfação booleana é um problema de referência em teoria de complexidade, porque o funcionamento de uma máquina não-determinística pode ser modelado através de uma expressão lógica (ver Teorema de Cook 7).

4.4 Grafos

Há muitos problemas que envolvem a construção de um grafo. Um exemplo é o problema da coloração de um mapa em que se pretende determinar o número mínimo de cores necessárias a colorir um mapa, de modo a que países adjacentes tenham cores diferentes. A resolução envolve a construção de um grafo em que se associa um vértice a cada país, existindo uma aresta entre dois países se eles forem adjacentes. Depois, a determinação do número mínimo de cores pode ser feita resolvendo um problema de coloração de vértices de um grafo (ver Exemplo).

Nesta secção, apresenta-se um conjunto de problemas definidos em grafos em que existem variáveis de decisão associadas a arestas ou vértices do grafo.

4.4.1 Cobertura (mínima) das arestas de um grafo

Dado um grafo $G = (V, A)$, em que V é conjunto de vértices e A o conjunto de arestas, o problema consiste em determinar o conjunto de vértices C de menor cardinal de modo a que qualquer aresta seja incidente, pelo menos, num dos vértices do conjunto C.

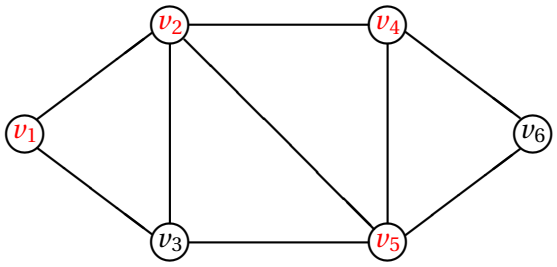
O problema pode ser formulado usando variáveis de decisão x_{ij}, cujo significado é o seguinte:

$$x_j = \begin{cases} 1 & \text{, se o vértice } j \text{ for seleccionado para o conjunto } C, \\ 0 & \text{, caso contrário.} \end{cases}$$

A formulação do problema é a seguinte:

$$\begin{aligned} \min z &= \sum_{i \in V} x_i \\ \text{suj. a} \quad & x_i + x_j \geq 1, \ \forall i, j \in V \\ & x_i = 0 \text{ ou } 1, \ \forall i \in V \end{aligned}$$

Exemplo 4.4. Considere o seguinte grafo em que as arestas representam os corredores de um museu. Um guarda colocado num vértice do grafo vigia todos corredores (as arestas) que são incidentes no vértice, e só esses. O objectivo é determinar o número mínimo de guardas que são necessários para vigiar todos os corredores do museu.



O modelo de programação linear é o seguinte:

```
 /* função objectivo */
min: x1 + x2 + x3 + x4+ x5 + x6;

 /*  Restrições  */
arestav1v2: x1 + x2 >=1;
arestav1v3: x1 + x3 >=1;
arestav2v3: x2 + x3 >=1;
arestav2v4: x2 + x4 >=1;
arestav2v5: x2 + x5 >=1;
arestav3v5: x3 + x5 >=1;
arestav4v5: x4 + x5 >=1;
arestav4v6: x4 + x6 >=1;
arestav5v6: x5 + x6 >=1;

// restrições de integralidade
bin x1, x2, x3, x4, x5, x6;
```

Uma das coberturas mínimas do grafo é o conjunto $C = \{v_1, v_2, v_4, v_5\}$.

□

4.4.2 Conjunto (máximo) de vértices independentes de um grafo

Dado um grafo $G = (V, A)$, em que V é conjunto de vértices e A o conjunto de arestas, o problema consiste em determinar o conjunto de vértices I de maior cardinal de modo a que qualquer aresta seja incidente num dos vértices do conjunto I.

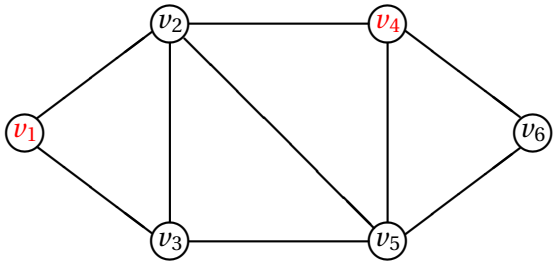
O problema pode ser formulado usando variáveis de decisão x_{ij}, cujo significado é o seguinte:

$$x_j = \begin{cases} 1 & \text{, se o vértice } j \text{ for seleccionado para o conjunto } I, \\ 0 & \text{, caso contrário.} \end{cases}$$

A formulação do problema é a seguinte:

$$\begin{aligned} \max z \quad &= \quad \sum_{j \in V} x_j \\ \text{suj. a} \quad & \quad x_i + x_j \geq 1, \ \forall i, j \in V \\ & \quad x_i = 0 \text{ ou } 1, \ \forall i \in V \end{aligned}$$

Exemplo 4.5. Considere o seguinte grafo em que arcos representam os corredores de um museu. Um guarda colocado num vértice do grafo vigia todos os arcos que são incidentes no vértice e vê os guardas (eventualmente) colocados nos vértices adjacentes, e só esses. O objectivo do problema é determinar o número máximo de guardas que podem ser colocados em vértices do grafo de modo a que nenhum guarda veja outro guarda.



O modelo de programação linear é o seguinte:

```
/* função objectivo */
max: x1 + x2 + x3 + x4+ x5 + x6;

 /*  Restrições  */
arestav1v2: x1 + x2 <=1;
arestav1v3: x1 + x3 <=1;
arestav2v3: x2 + x3 <=1;
arestav2v4: x2 + x4 <=1;
arestav2v5: x2 + x5 <=1;
arestav3v5: x3 + x5 <=1;
arestav4v5: x4 + x5 <=1;
arestav4v6: x4 + x6 <=1;
arestav5v6: x5 + x6 <=1;

// restrições de integralidade
bin x1, x2, x3, x4, x5, x6;
```

Um dos conjuntos máximos de vértices independentes é $I = \{v_1, v_4\}$. □

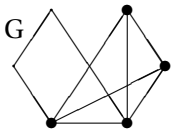
4.4.3 Clique (máxima) de um grafo

Dado um grafo $G = (V, A)$, em que V é conjunto de vértices e A o conjunto de arestas, o problema consiste em determinar o conjunto de vértices C de maior cardinal de modo a que haja uma aresta

entre cada par de vértices do conjunto C.

Um grafo em que exista uma aresta entre cada par de vértices designa-se por grafo completo ou clique. O problema é equivalente a determinar o maior subgrafo completo existente em G.

Exemplo 4.6. Considere o seguinte grafo, em que os vértices representam pessoas e uma aresta indica que duas pessoas são compatíveis. Sabendo que é necessário formar uma equipa em que todas as pessoas sejam compatíveis entre si, determinar qual o cardinal do maior equipa que se pode formar.



A clique máxima do grafo G tem cardinal 4. □

Este problema é equivalente ao problema de vértices independentes de um grafo, como se mostra na secção A.5.4.

4.4.4 Coloração dos vértices de um grafo

Dado um grafo $G = (V, A)$, em que V é conjunto de vértices e A o conjunto de arestas, o problema consiste em determinar o número mínimo de cores necessárias para colorir os vértices do grafo de modo a que dois vértices adjacentes tenham cores diferentes. Seja $n = |V|$ o cardinal do conjunto de vértices. Pode ser necessário usar n cores diferentes, se existir uma aresta entre cada par de vértices. Um grafo com essas características designa-se por *grafo completo*.

A formulação do problema de coloração de vértices abaixo apresentada é baseada em variáveis de decisão x_{ij}, cujo significado é o seguinte:

$$x_{ik} = \begin{cases} 1 & \text{, se o vértice } i \text{ é tiver a cor } k \\ 0 & \text{, caso contrário} \end{cases}$$

Quando uma variável x_{ik} assume um valor positivo, isso significa que cor k é utilizada. As variáveis y_k contam o número de cores usadas:

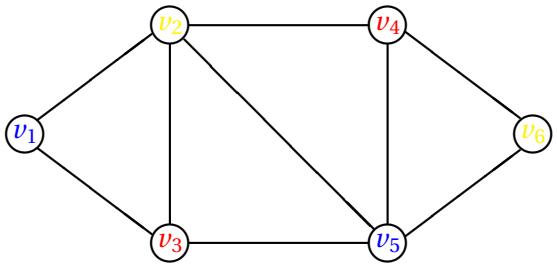
$$y_k = \begin{cases} 1 & \text{, se a cor } k \text{ é usada} \\ 0 & \text{, caso contrário} \end{cases}$$

A formulação do problema é a seguinte:

$$\begin{aligned} \min z &= \sum_{i \in V} y_k \\ \text{suj. a} \quad & \sum_{k=1}^{n} x_{ik} = 1, \ \forall i \in V \\ & x_{ik} \leq y_k, \ \forall i \in I, k = 1, \ldots, n \\ & x_{ik} + x_{jk} \leq 1, \ \forall i, j \in V, k = 1, \ldots, n \\ & x_{ik} = 0 \text{ ou } 1, \ \forall i \in V, k = 1, \ldots, n \\ & y_k = 0 \text{ ou } 1, \ k = 1, \ldots, n. \end{aligned}$$

O primeiro grupo de restrições determina que seja usada exactamente uma das cores para colorir o vértice i. O segundo grupo de restrições impõe que se use uma cor se algum vértice a usar. O terceiro grupo de restrições determina que, se o vértice i e o vértice j forem adjacentes (unidos por um aresta), eles não poderão ter a mesma cor k.

Exemplo 4.7. Considere o seguinte grafo em que arcos representam os corredores de um museu. Um guarda colocado num vértice do grafo vigia todos os arcos que são incidentes no vértice e vê os guardas (eventualmente) colocados nos vértices adjacentes, e só esses. Sabendo que é necessário haver guardas em todos os vértices do grafo, determinar o número mínimo de cores de fardas de modo a que cada guarda só veja guardas com fardas de cores diferentes da sua.



O modelo de programação linear é o seguinte:

```
/* função objectivo */
min: y1 + y2 + y3 + y4+ y5 + y6;

// cada vértice deve ter uma cor:
// apenas uma das variáveis pode assumir o valor 1 para cada vértice
v1: x11 + x12 + x13 + x14 + x15 + x16 = 1;
v2: x21 + x22 + x23 + x24 + x25 + x26 = 1;
v3: x31 + x32 + x33 + x34 + x35 + x36 = 1;
v4: x41 + x42 + x43 + x44 + x45 + x46 = 1;
v5: x51 + x52 + x53 + x54 + x55 + x56 = 1;
v6: x61 + x62 + x63 + x64 + x65 + x66 = 1;

/* se um vértice usar uma cor, então essa cor deve ser usada */
cor1: x11 <= y1; x21 <= y1; x31 <= y1; x41 <= y1; x51 <= y1; x61 <= y1;
cor2: x12 <= y2; x22 <= y2; x32 <= y2; x42 <= y2; x52 <= y2; x62 <= y2;
cor3: x13 <= y3; x23 <= y3; x33 <= y3; x43 <= y3; x53 <= y3; x63 <= y3;
cor4: x14 <= y4; x24 <= y4; x34 <= y4; x44 <= y4; x54 <= y4; x64 <= y4;
cor5: x15 <= y5; x25 <= y5; x35 <= y5; x45 <= y5; x55 <= y5; x65 <= y5;
cor6: x16 <= y6; x26 <= y6; x36 <= y6; x46 <= y6; x56 <= y6; x66 <= y6;

/*  se o vértice i e o vértice j forem adjacentes (unidos por uma aresta),
 não poderão ter a mesma cor k   */
v1v3: x11+x31<=1; x12+x32<=1; x13+x33<=1; x14+x34<=1; x15+x35<=1; x16+x36<=1;
v2v3: x21+x31<=1; x22+x32<=1; x23+x33<=1; x24+x34<=1; x25+x35<=1; x26+x36<=1;
```

```
v2v4: x21+x41<=1; x22+x42<=1; x23+x43<=1; x24+x44<=1; x25+x45<=1; x26+x46<=1;
v2v5: x21+x51<=1; x22+x52<=1; x23+x53<=1; x24+x54<=1; x25+x55<=1; x26+x56<=1;
v3v5: x31+x51<=1; x32+x52<=1; x33+x53<=1; x34+x54<=1; x35+x55<=1; x36+x56<=1;
v4v5: x41+x51<=1; x42+x52<=1; x43+x53<=1; x44+x54<=1; x45+x55<=1; x46+x56<=1;
v4v6: x41+x61<=1; x42+x62<=1; x43+x63<=1; x44+x64<=1; x45+x65<=1; x46+x66<=1;
v5v6: x51+x61<=1; x52+x62<=1; x53+x63<=1; x54+x64<=1; x55+x65<=1; x56+x66<=1;

// restrições de integralidade
bin x11, x12, x13, x14, x15, x16;
bin x21, x22, x23, x24, x25, x26;
bin x31, x32, x33, x34, x35, x36;
bin x41, x42, x43, x44, x45, x46;
bin x51, x52, x53, x54, x55, x56;
bin x61, x62, x63, x64, x65, x66;
bin y1, y2, y3, y4, y5, y6;
```

Uma das soluções óptimas do problema é $y_1 = y_2 = y_3 = 1$ e $x_{11} = x_{23} = x_{32} = x_{42} = x_{51} = x_{63} = 1$, sendo as restantes variáveis iguais a 0, o que significa que são usadas três cores, sendo os vértices v_1 e v_5 pintados com a cor 1, os vértices v_3 e v_4 pintados com a cor 2, e os vértices v_2 e v_6 pintados com a cor 3. □

4.4.5 Emparelhamento máximo

Dado um grafo $G = (V, A)$, em que V é conjunto de vértices e A o conjunto de arestas, o problema consiste em determinar o conjunto de arestas E de maior cardinal de modo a que qualquer vértice tenha, no máximo, uma aresta incidente.

O problema pode ser formulado usando variáveis de decisão x_e, cujo significado é o seguinte:

$$x_{ij} = \begin{cases} 1 & \text{, se a aresta } (i,j) \text{ for seleccionada para o conjunto } E, \\ 0 & \text{, caso contrário.} \end{cases}$$

A formulação do problema é a seguinte:

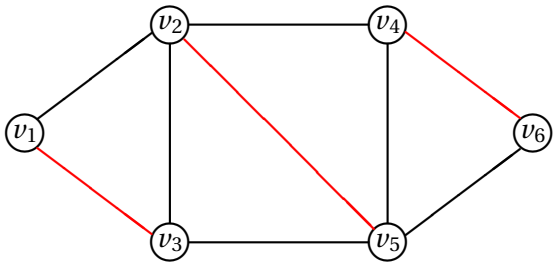
$$\max \quad \sum_{(i,j)\in A} x_{ij} \tag{4.1}$$

$$\text{suj. a} \quad \sum_{(i,j)\in F(i)} x_{ij} \leq 1 \ , \ \forall i \in V \tag{4.2}$$

$$x_{ij} \text{ binário} \ , \ \forall (i,j) \in A \tag{4.3}$$

sendo $F(i) = \{(i,j) \in A : \text{uma extremidade da aresta é incidente no vértice } i\}$. Assim, o primeiro conjunto de restrições garante que, de todas as arestas incidentes no vértice i, apenas um pode pertencer ao emparelhamento.

Exemplo 4.8. Considere o seguinte grafo em que vértices representam pessoas e um arco entre duas pessoas significa que elas são compatíveis, podendo formar uma equipa. O objectivo do problema é determinar o número máximo de equipas.



O modelo de programação linear é o seguinte:

```
 /* função objectivo */
max: x12 + x13 + x23 + x24 + x25 + x35 + x45 + x46 + x56;

 /* Restrições */
v1: x12 + x13 <=1;
v2: x12 + x23 + x24 + x25 <=1;
v3: x13 + x23 + x35 <=1;
v4: x24 + x45 + x46 <=1;
v5: x25 + x35 + x45 + x56 <=1;
v6: x46 + x56 <=1;

// restrições de integralidade
bin x12, x13, x23, x24, x25, x35, x45, x46, x56;
```

Um dos emparelhamentos máximos do grafo é o conjunto de arestas $E = \{(1,3),(2,5),(4,6)\}$. □

4.5 Selecção de projectos de investimento

Nos problemas de selecção de projectos de investimento pretende-se determinar quais os projectos a escolher de entre um conjunto de projectos alternativos, de modo a optimizar uma dada medida de eficiência. É comum, nestes problemas, haver relações de mútua exclusão entre os projectos, de implicação de escolha e relações de outros tipos, que é necessário traduzir em relações lógicas.

Tipicamente, estes problemas são formulados como problemas de programação inteira, associando uma variável binária a cada projecto:

$$x_j = \begin{cases} 1 & \text{, se o projecto } j \text{ é seleccionado} \\ 0 & \text{, caso contrário} \end{cases}$$

As relações entre projectos são explicitadas através de restrições envolvendo as variáveis binárias.

Exemplo 4.9. Considere 4 projectos de investimento com os cashflows indicados no Quadro 4.1. Foi definido que se deveria procurar o plano de investimento que, respeitando as condições impostas, maximizasse o valor presente, um dos critérios habitualmente usados. Usando as fórmulas

	Cashflow do Projecto			
ano	A	B	C	D
0	-500	-250	-400	-100
1	-600	-100	50	-40
2	200	150	200	20
3	500	350	200	180
4	700	0	100	100
5	300	0	100	0
valor presente (i=15%)	7.6	6.6	33.1	55.9

Tabela 4.1: Projectos de investimento

da matemática financeira, com uma taxa de juro interna de 15%, foram determinados os valores presentes de cada projecto, que são os apresentados no Quadro.

As restrições a cumprir são as seguintes:

- apenas um dos projectos B ou C pode ser seleccionado,
- a selecção do projecto D implica a selecção do projecto A,
- no ano 0, a companhia dispõe de 850 U.M., e
- no ano 1, a companhia dispõe de 750 U.M..

Como regra geral, qualquer expressão lógica pode ser expressa como uma restrição envolvendo variáveis binárias. Os valores lógicos "verdadeiro"e "falso"correspondem, respectivamente, aos valores 1 e 0 das variáveis binárias. As restrições que se seguem servem para ilustrar a modelação de condições lógicas.

Podemos exprimir a condição de apenas um dos projectos B ou C poder ser seleccionado, ou então, nenhum deles, através da restrição:

$$x_B + x_C \leq 1.$$

Por outro lado, a selecção do projecto D implica a selecção do projecto A. Esta condição pode ser expressa do seguinte modo:

$$x_A \geq x_D \text{ ou seja } x_D - x_A \leq 0.$$

As limitações de capital para investir no ano 0 e no ano 1 podem ser traduzidas da seguinte forma, respectivamente:

$$\begin{aligned} 500x_A + 250x_B + 400x_C + 100x_D &\leq 850 \\ 600x_A + 100x_B - 50x_C + 40x_D &\leq 750 \end{aligned}$$

Saliente-se que, no ano 1, o projecto x_C já tem cashflow positivo, pelo que, a ser seleccionado, pode aumentar o capital disponível para investimento esse ano.

A função objectivo de maximização do valor presente corresponde a:

$$\max 7.6x_A + 6.6x_B + 33.1x_C + 55.9x_D.$$

□

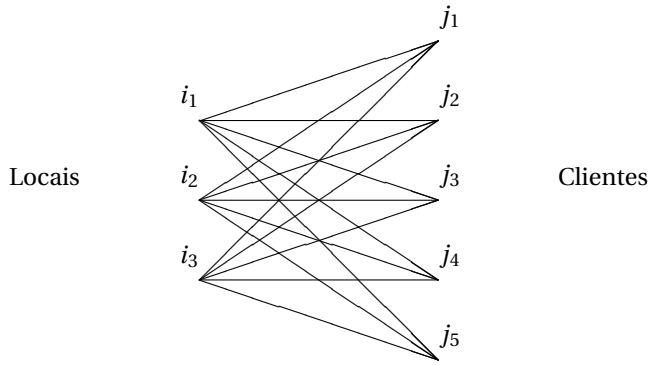


Figura 4.1: Localização de instalações

4.6 Localização de instalações

O problema de localização consiste em seleccionar os locais onde devem ser estabelecidas instalações (por exemplo, armazéns) de forma a melhor servir um dado conjunto de clientes. Existe um conjunto de m possíveis locais disponíveis, designado por I, onde podem ser estabelecidas as instalações, e um conjunto de n clientes, designado por J, que é necessário servir.

Há muitas variantes deste problema com diferentes tipos de função objectivo e de restrições. O objectivo pode ser minimizar custos totais de operação do sistema, como, por exemplo, em sistemas de distribuição de mercadorias, ou minimizar a maior distância que é necessário percorrer para atingir qualquer cliente, situação que ocorre em serviços de emergência, como, por exemplo, hospitais ou bombeiros.

Iremos apresentar a versão não-capacitada deste problema, usualmente designada, na literatura anglo-saxónica, por *uncapacitated facility location problem*. Nesta versão, não há valores de capacidade associados às instalações, pelo que qualquer instalação pode fornecer as quantidades requeridas por qualquer conjunto de clientes, ou até, a soma de todas as quantidades de todos os clientes.

Iremos também supor que não existe nenhuma imposição sobre o número de instalações a seleccionar, podendo esse número ser igual ao número de locais disponíveis. Neste problema, é necessário encontrar um balanceamento entre os custos de transporte e os custos fixos das instalações. Um maior número de instalações pode reduzir os custos de transporte, mas acarreta maiores custos fixos.

Exemplo 4.10. A Figura 4.1 ilustra uma situação com três possíveis localizações para as instalações e cinco clientes.

□

Os custos unitários de transporte entre o local i e o cliente j são designados por c_{ij}. A título de exemplo, estes custos podem representar, para um dado período de tempo, custos de deslocação para abastecimento de clientes ou custos de utilização de linhas de telecomunicação. Por outro lado, associado ao estabelecimento de cada instalação, existe um custo fixo designado por f_i, que representa uma renda de utilização.

O problema pode ser formulado como um modelo de programação matemática usando as se-

guintes variáveis de decisão:

$$x_{ij} = \begin{cases} 1 & \text{, se o cliente } j \text{ está associado ao local } i \\ 0 & \text{, caso contrário} \end{cases}$$

e variáveis de decisão binárias, y_i, associadas a cada local:

$$y_i = \begin{cases} 1 & \text{, se o local } i \text{ é seleccionado} \\ 0 & \text{, caso contrário} \end{cases}$$

4.6.1 Formulação I

Uma formulação matemática do problema de localização não-capacitado é a seguinte:

$$\begin{aligned}
\min \quad & \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} + \sum_{i\in I} f_i y_i && (4.4)\\
\text{suj. a} \quad & \sum_{i\in I} x_{ij} = 1\ ,\ \forall j && (4.5)\\
& \sum_{j=1}^{n} x_{ij} \le n y_i, \forall i && (4.6)\\
& x_{ij} \text{ binário}, \forall i, j && (4.7)\\
& y_i \text{ binário}, \forall i && (4.8)
\end{aligned}$$

O primeiro grupo de restrições força que cada cliente j seja afecto apenas a um local i. Quando um cliente j é servido a partir de uma instalação i, o respectivo custo de abrir essa instalação, f_i, deve ser imputado aos custos totais. Num problema com n clientes, uma forma de exprimir essa implicação, para cada local i, é a apresentada no segundo grupo de restrições: qualquer variável x_{ij} que tome o valor 1 força a variável y_i correspondente a ser também igual a 1.

Esta formulação das restrições de implicação conduz a uma relaxação linear muito pobre.

4.6.2 Formulação II

Uma outra forma de obter o mesmo conjunto de soluções 0-1 para o problema de localização não-capacitado é através da seguinte formulação:

$$\begin{aligned}
\min \quad & \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} + \sum_{i\in I} f_i y_i && (4.9)\\
\text{suj. a} \quad & \sum_{i\in I} x_{ij} = 1\ ,\ \forall j && (4.10)\\
& y_i - x_{ij} \ge 0\ ,\ \forall i, j && (4.11)\\
& x_{ij} \text{ binário}, \forall i, j && (4.12)\\
& y_i \text{ binário}, \forall i && (4.13)
\end{aligned}$$

que corresponde a desagregar as restrições de implicação. O segundo grupo de restrições força o estabelecimento de uma instalação no local i sempre que o cliente j está associado a essa instalação, *i.e.*, se alguma quantidade x_{ij} for positiva, a variável binária y_i correspondente deve tomar o valor 1. Em sentido contrário, quando uma variável y_i tome o valor 0, todas as quantidades x_{ij} devem ser nulas. Embora o número de restrições da formulação seja muito mais elevado, este formulação é preferível, por ter uma relaxação linear mais forte.

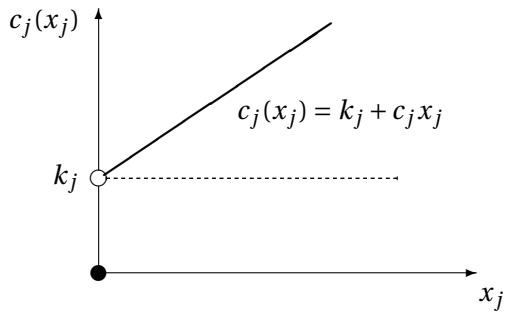


Figura 4.2: Função custo de produção não-linear

4.7 Custo fixo

O custo de produção de um lote de artigos é uma função não-linear, composta por um custo fixo de preparação e custos variáveis que dependem do número de unidades produzidas. Para um artigo j, a função custo, designada por $c_j(x_j)$, traduz os custos incorridos quando se produzem x_j artigos do tipo j. Esta função, apresentada na Figura 4.2, é a seguinte:

$$c_j(x_j) = \begin{cases} k_j + c_j x_j & \text{, se } x_j > 0 \\ 0 & \text{, se } x_j = 0 \end{cases}$$

É possível modelar funções não-lineares utilizando variáveis de decisão binárias y_j, associadas a cada artigo do tipo j, com os seguintes valores:

$$y_j = \begin{cases} 1 & \text{, se o artigo } j \text{ é produzido} \\ 0 & \text{, caso contrário} \end{cases}$$

Um problema de planeamento da produção em que os custos de produção de um artigo fossem conforme os acima descritos poderia ser formulado do seguinte modo:

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} (c_j x_j + k_j y_j) \\ \text{suj. a} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \ i = 1,2,\ldots,m \\ & x_j \leq M y_j, \ j = 1,2,\ldots,n \\ & x_j \geq 0 \text{ e inteiro}, \ j = 1,2,\ldots,n \end{aligned}$$

sendo M uma quantidade suficientemente grande, por exemplo, maior que a dimensão do maior lote que pode ser produzido. As restrições asseguram que, sempre que $x_j > 0$, a variável de decisão binária $y_j = 1$, forçando o respectivo custo fixo k_j a ser incluído na função objectivo. Naturalmente, quando $x_j = 0$, a variável binária y_j assume o valor nulo, por essa opção ser mais favorável para um problema de minimização de custos. O conjunto de restrições do tipo $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$ expressam outro tipo de limitações gerais existentes no sistema.

Este problema envolve variáveis de decisão binárias e variáveis de decisão contínuas, sendo usualmente designado por problema de programação inteira misto.

Exemplo 4.11. Em primeiro lugar, vamos ver um modelo apenas com custos variáveis (lineares), cujo objectivo é decidir quantos artigos produzir de cada tipo (o número total deve ser 100), obedecendo a uma restrição de capacidade, de modo a minimizar os custos de produção.

$$\begin{array}{ll} \min & 4x_1 + 3x_2 + 4x_3 \\ \text{suj. a} & 1x_1 + 1x_2 + 1x_3 = 100 \\ & 4x_1 + 6x_2 + 5x_3 \leq 480 \\ & x_j \geq 0 \text{ e inteiro, } j = 1,2,3 \end{array}$$

A solução óptima é produzir 60 artigos de tipo 1 e 40 artigos de tipo 2, com um custo óptimo de 360.

□

Exemplo 4.12. Uma outra versão deste modelo, além dos custos variáveis, usa com custos fixos, que resultam da preparação das máquinas quando há mudança de produção de artigo. O custo de preparação da máquina é igual a 50, qualquer que seja o tipo de artigo:

$$c_1(x_1) = \begin{cases} 4x_1 + 50 & \text{, se } x_1 > 0 \\ 0 & \text{, se } x_1 = 0 \end{cases}$$

$$c_2(x_2) = \begin{cases} 3x_2 + 50 & \text{, se } x_2 > 0 \\ 0 & \text{, se } x_2 = 0 \end{cases}$$

$$c_3(x_3) = \begin{cases} 4x_3 + 50 & \text{, se } x_3 > 0 \\ 0 & \text{, se } x_3 = 0 \end{cases}$$

A existência de custos fixos de preparação favorece a produção de um menor número de tipos de artigos.

$$\begin{array}{ll} \min & 4x_1 + 3x_2 + 4x_3 + 50y_1 + 50y_2 + 50y_3 \\ \text{suj. a} & 1x_1 + 1x_2 + 1x_3 = 100 \\ & 4x_1 + 6x_2 + 5x_3 \leq 480 \\ & x_1 \leq My_1 \\ & x_2 \leq My_2 \\ & x_3 \leq My_3 \\ & x_j \geq 0 \text{ e inteiro, } j = 1,2,3 \\ & y_j \text{ binário, } j = 1,2,3 \end{array}$$

O valor de M pode ser igual a 100. A solução óptima é produzir 100 artigos de tipo 1, com um custo óptimo de 450.

□

4.8 Restrições activas e redundantes

Uma restrição pode ser tornada activa ou redundante usando variáveis binárias. Adicionando ao lado direito da restrição $A^1 x \leq b^1$ (do conjunto de restrições, $Ax \leq b$) o termo $M(1-y_1)$, que é uma função da variável binária y_1, obtém-se:

$$A^1 x \leq b^1 + M(1-y_1)$$

A restrição original $A^1 x \leq b^1$ é *activa* (deve ser obedecida), se $y_1 = 1$, ou *redundante* (não precisa de ser obedecida), se $y_1 = 0$. M é uma constante com um valor suficientemente grande para que não seja eliminada nenhuma solução admissível noutras restrições quando a restrição $A^1 x \leq b^1$ for redundante.

A selecção de k restrições activas é um exemplo. Dadas m restrições $A^i x \leq b^i,\ i = 1,2,\ldots,m$, podemos impor que apenas k das m restrições sejam activas:

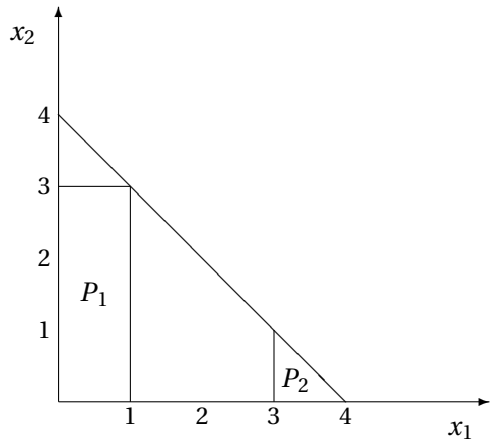
$$\begin{aligned} & A^i x \leq b^i + M(1-y_i),\ i = 1,\ldots,m \\ & \sum_{i=1}^{m} y_i = k \\ & y_i \text{ binário},\ i = 1,\ldots,m \end{aligned}$$

O mesmo se aplica se, em vez de termos m restrições, tivermos m conjuntos de restrições $P_i = \{x : A^p x \leq b^p, x \geq 0\},\ i = 1,\ldots,m$, em que A^p representa uma matriz com p linhas e b^p um vector com p linhas.

4.8.1 União de domínios convexos

Relembra-se que a **conjunção** de restrições lineares define sempre um domínio convexo. O domínio resultante é a **intersecção** de domínios convexos, que é um domínio convexo. Os domínios não-convexos podem ser modelados usando programação inteira. Uma **conjunção** de restrições lineares, em que o uso de variáveis binárias permite tornar as restrições activas ou redundantes, define afinal uma **disjunção** de restrições ou de grupos de restrições. O domínio resultante é a **união** de domínios convexos, que pode não ser convexo.

Exemplo 4.13. O domínio indicado na Figura é uma *Dicotomia,* constituída por dois domínios convexos P_1 e P_2, cada um deles definido através de um conjunto de restrições lineares. Os pontos admissíveis são os que obedecem a um dos conjuntos de restrições. O domínio resultante é não-convexo, e representa a união dos 2 domínios convexos.



Os domínios convexos P_1 e P_2 são definidos do seguinte modo, respectivamente:

$$P_1 : \begin{cases} x_1 \leq 1 \\ x_2 \leq 3 \\ x_1, x_2 \geq 0 \end{cases} \qquad P_2 : \begin{cases} x_1 \geq 3 \\ x_1 + x_2 \leq 4 \\ x_1, x_2 \geq 0 \end{cases}$$

O domínio da dicotomia são os pontos que obedecem a um dos conjuntos de restrições. Quando y_1 é igual a 1, as restrições que definem o domínio P_1 são activas e as restantes que definem o domínio P_2 são redundantes. Exactamente umas das variáveis y_1 e y_2 deve ser igual a 1. Quando $y_2 = 1$, passa-se contrário. O conjunto de restrições com variáveis binárias que define o domínio pretendido é o seguinte:

$$\begin{aligned} x_1 &\leq 1 + M(1 - y_1) \\ x_2 &\leq 3 + M(1 - y_1) \\ -x_1 &\leq -3 + M(1 - y_2) \\ x_1 + x_2 &\leq 4 + M(1 - y_2) \\ y_1 + y_2 &= 1 \\ x_1, x_2 \geq 0 \quad &, \quad y_i \text{ binários} \end{aligned}$$

o valor de M deve ser suficientemente grande para não eliminar nenhuma solução que se pretenda seja admissível. Pode substituir-se y_1 por y, e y_2 por $(1 - y)$.

□

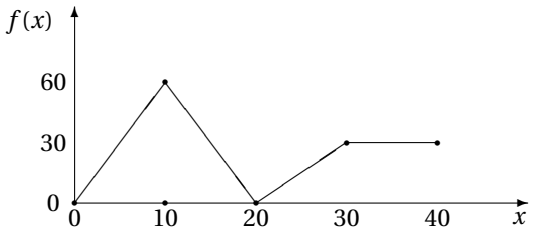
4.9 Função linear por partes

Podemos estar interessados em usar funções, cuja definição analítica é um conjunto de funções lineares por partes.

Exemplo 4.14. Considere a seguinte função:

$$f(x) = \begin{cases} 6x & \text{, se } 0 \le x \le 10 \\ 120 - 6x & \text{, se } 10 \le x \le 20 \\ -60 + 3x & \text{, se } 20 \le x \le 30 \\ 30 & \text{, se } 30 \le x \le 40 \end{cases}$$

cuja representação gráfica é a seguinte:



Há várias formas de modelar funções lineares por partes, uma das quais se apresenta de seguida.

$$\begin{aligned} f(x) &= (0y_1 + 6x_1) + (120y_2 - 6x_2) + (-60y_3 + 3x_3) + (30y_4 + 0x_4) \\ 1 &= y_1 + y_2 + y_3 + y_4 \\ x &= x_1 + x_2 + x_3 + x_4 \\ 0y_1 &\le x_1 \le 10y_1 \\ 10y_2 &\le x_2 \le 20y_2 \\ 20y_3 &\le x_3 \le 30y_3 \\ 30y_4 &\le x_4 \le 40y_4 \\ & x_i \ge 0 \text{ e inteiro}, i = 1, \ldots, 4 \\ & y_i \in \{0,1\}, i = 1, \ldots, 4 \end{aligned}$$

□

4.10 Problemas de planeamento numa única máquina

Nos problemas de planeamento numa única máquina, é necessário determinar a sequenciação das tarefas que minimiza uma dada medida de eficiência. As tarefas são definidas por um tempo de processamento p_j. Vamos associar a cada tarefa uma variável de decisão x_j, que significa o instante de início da execução da tarefa j.

4.10.1 Restrições de não-simultaneidade

As dicotomias podem ser também utilizadas em problemas de planeamento para exprimir o facto de não poder haver duas tarefas a ocupar simultaneamente uma máquina. Dado um par de tarefas i e j, podemos garantir que elas não ocupam simultaneamente a máquina, se o instante do fim da execução de qualquer uma delas for anterior ao instante de início da outra. Isto pode ser traduzido através de duas restrições disjuntivas, que são:

$$x_i + p_i \leq x_j \quad \text{ou} \quad x_j + p_j \leq x_i.$$

Usando apenas uma variável y_{ij} para exprimir a dicotomia:

$$y_{ij} = \begin{cases} 1 & \text{, se a tarefa } i \text{ precede a tarefa } j \\ 0 & \text{, se a tarefa } j \text{ precede a tarefa } i \end{cases}$$

as restrições de não-simultaneidade podem ser expressas por:

$$x_i + p_i \leq x_j + M(1 - y_{ij})$$
$$x_j + p_j \leq x_i + M y_{ij}$$

4.10.2 Outras restrições

Nos problemas de planeamento numa única máquina, poderá haver outras restrições dos seguintes tipos:

- datas de disponibilidade: se uma tarefa só puder ser executada a partir de uma data r_j, podemos usar $x_j \geq r_j$.
- prazos de entrega: se uma tarefa tiver que ser concluída antes da data limite de entrega d_j, podemos usar $x_j + p_j \leq d_j$.
- precedências: se uma tarefa i tiver que ser executada obrigatoriamente antes da tarefa j, as restrições disjuntivas de não-simultaneidade devem ser substituídas por $x_i + p_i \leq x_j$.

4.10.3 Medidas de eficiência

Há também a considerar vários tipos de medida de eficiência do sistema, que é necessário traduzir numa função objectivo. Vamos designar por $C_j = x_j + p_j$ o instante em que termina a execução da tarefa. As medidas de eficiência mais usadas são:

- $C_{max} = max_j\ C_j$: minimização do tempo de execução de todas as tarefas, ou seja, o instante em que termina a execução da última tarefa. O instante C_{max} é usualmente designado por *makespan*.
- $L_{max} = max_j(C_j - d_j)$: minimização do maior atraso (*maximum lateness*) existente numa das tarefas.
- $T = \sum_{i=1}^{n} T_j$: sendo $T_j = max[0, C_j - d_j]$, o atraso positivo (*tardiness*): minimização da soma dos atrasos das tarefas que terminam atrasadas.
- $F = \sum_{i=1}^{n} C_j$: minimização da soma dos tempos de permanência das tarefas no sistema (medida equivalente ao tempo médio de permanência, porque o número de tarefas é constante).
- $U = \sum_{i=1}^{n} U_j$, sendo $U_j = \begin{cases} 1 & \text{, se } C_j > d_j \\ 0 & \text{, caso contrário} \end{cases}$:

  minimização do número de tarefas em atraso, independentemente do valor do seu atraso.

Nas medidas de eficiência T, F e U, é comum ainda associar coeficientes de ponderação w_j às tarefas, que traduzem a sua importância relativa. As medidas de eficiência passam a ser definidas como uma soma ponderada.

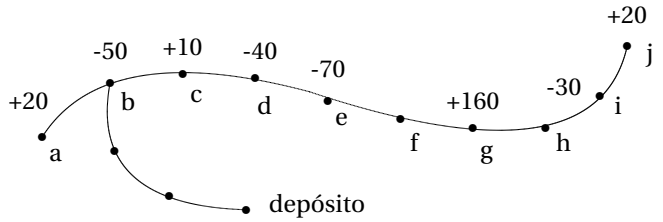
Capítulo 5

Aplicações

5.1 Transporte de terra

As obras de terraplanagem representam uma parte significativa dos custos de construção de vias de comunicação. Grandes volumes de terra devem ser deslocados de zonas de empréstimo para zonas de depósito para obter os nivelamentos desejados. Os custos de transporte de terra são aproximadamente proporcionais à distância percorrida.

Considere a estrada representada na figura. Os pontos referidos por $a, b, \ldots, j$ encontram-se distanciados entre si de 10 Km. As quantidades associadas a estes pontos indicam os volumes de terra a deslocar (em milhares de m^3), sendo as zonas de empréstimo e de depósito assinaladas pelos sinais + e −, respectivamente. Em caso de necessidade, pode ainda recorrer-se a uma zona de depósito, situada fora do traçado da via, a uma distância de 30 Km do ponto b.



O objectivo é determinar os movimentos de terra a efectuar, de modo a minimizar os custos de terraplanagem.

O modelo de programação linear é o seguinte:

```
/*
      TRANSPORTE DE TERRAS

      Xpq = volume de terra a transportar [em m^3]
             da origem p para o destino q

      p \in {a,c,g,j}
      q \in {b,d,e,i,Dep}

      Cpq = custo de transporte (distância) [em U.M / m^3 ]
```

```
            entre a origem p e o destino q
*/

min: 10 Xab + 30 Xad + 40 Xae + 80 Xai +  40 XaDep
   + 20 Xcb + 10 Xcd + 20 Xce + 60 Xci +  40 XcDep
   + 50 Xgb + 30 Xgd + 20 Xge + 20 Xgi +  80 XgDep
   + 80 Xjb + 60 Xjd + 50 Xje + 10 Xji + 110 XjDep;

//   a soma dos volumes de terra que saem de uma dada zona de empréstimo
//   para cada zona de depósito perfaz o volume a deslocar

      Xab +   Xad +   Xae +   Xai +   XaDep =  20 ; // (saem da zona a)
      Xcb +   Xcd +   Xce +   Xci +   XcDep =  10 ; // (saem da zona c)
      Xgb +   Xgd +   Xge +   Xgi +   XgDep = 160 ; // (saem da zona g)
      Xjb +   Xjd +   Xje +   Xji +   XjDep =  20 ; // (saem da zona j)

//   a soma dos volumes de terra que chegam a uma dada zona de depósito
//   de cada zona de empréstimo perfaz o volume a depositar;

      Xab +   Xcb +   Xgb +   Xjb   = 50 ;  // (chegam à zona b)
      Xad +   Xcd +   Xgd +   Xjd   = 40 ;  // (chegam à zona d)
      Xae +   Xce +   Xge +   Xje   = 70 ;  // (chegam à zona e)
      Xai +   Xci +   Xgi +   Xji   = 30 ;  // (chegam à zona i)
      XaDep + XcDep + XgDep + XjDep = 20 ;  // (chegam ao Depósito)
```

A solução óptima do problema consiste nos seguintes movimentos de terra:

Variables	result
	5900
Xge	70
Xgb	40
Xgd	40
Xji	20
Xab	10
Xgi	10
XcDep	10
XaDep	10
Xad	0
Xae	0
Xai	0
Xcb	0
Xcd	0
Xce	0
Xci	0
XgDep	0
Xjb	0
Xjd	0
Xje	0
XjDep	0

5.2 Gestão de projectos: método do caminho crítico

O método do caminho crítico, designado na literatura anglo–saxónica por *critical path method (CPM)*, constitui uma ferramenta muito importante em gestão de projectos. O método do caminho crítico é aplicado a projectos que podem ser decompostos num conjunto de actividades, entre as quais existem relações de precedência. Todas as actividades têm de ser realizadas, e considera-se que têm durações determinísticas. As restrições de precedência traduzem o facto de o instante em

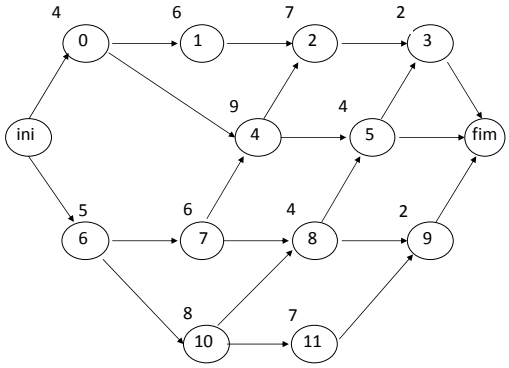
que se pode dar início a uma dada actividade ter de ser posterior aos instantes em que terminam as actividades que lhe são precedentes.

No método do caminho crítico, a rede que representa o projecto pode ser representada de duas formas alternativas: uma, em que as actividades do projecto são representadas por arcos do grafo, e a outra, em que são representadas por nós. Vamos considerar a segunda representação.

Considere um projecto com as actividades e as relações de precedência a seguir indicadas:

Actividade	Duração	Precedências
0	4	–
1	6	0
2	7	1,4
3	2	2,5
4	9	0,7
5	4	4,8
6	5	–
7	6	6
8	4	7,10
9	2	8,11
10	8	6
11	7	10

O grafo associado a este projecto, que inclui as actividades fictícias `ini` e `fim`, que podem ser pensadas como actividades, sem duração, de início e de conclusão do projecto, servindo para definir um vértice origem e um vértice destino do grafo, é o seguinte:



No problema em análise, o caminho crítico corresponde às actividades 6, 7, 4, 2 e 3, com uma duração de 29 unidades de tempo, que é também o menor tempo necessário para completar a execução de todo o projecto.

Caminho mais longo

O caminho crítico corresponde ao caminho mais longo entre o vértice que define o início do projecto e o vértice que define o fim do projecto. As actividades pertencentes ao caminho mais longo (que pode não ser único) são críticas, porque qualquer atraso verificado numa delas dá origem inevitavelmente a um atraso no tempo de execução do projecto global. Assim, estas actividades são as que devem ser seguidas e controladas mais de perto.

O problema de caminho mais longo entre um vértice s e um vértice t de um grafo acíclico (sem ciclos) pode ser formulado como um problema de programação linear utilizando variáveis de decisão x_{ij} associadas a cada arco (i, j) do grafo. Estas variáveis tomam o valor 1, se o arco fizer parte do caminho mais longo, e o valor 0, caso contrário.

O problema pode ser entendido como um modelo em que se injecta na rede, a partir do vértice s, uma unidade de fluxo que segue pelo caminho mais longo até atingir o vértice t. As restrições do problema, relativas a cada um dos vértices do grafo, traduzem a conservação de fluxo em cada vértice, ou seja, o número de unidades de fluxo que entram no vértice j através dos arcos (i, j) deve ser igual ao número de unidades que dele saem através dos arcos (j, k). Isto é equivalente a afirmar que, se há um arco do caminho a entrar num vértice, deverá haver um arco a sair; também que, se não houver arco a entrar no vértice, o caminho não passará pelo vértice. Esta relação é válida para todos os vértices, excepção feita ao vértice s, onde é introduzida uma unidade de fluxo na rede, e ao vértice t, onde a unidade é removida. De facto, a restrição relativa ao vértice t pode ser retirada do modelo, por ser linearmente dependente das outras restrições.

```
//   caminho mais longo num grafo acíclico
max: 4 x01 + 4 x04 + 6 x12 + 7 x23 + 2 x3f + 9 x42 + 9 x45 + 4 x53 +
      4 x5f + 5 x67 + 5 x610 + 6 x74 + 6 x78 + 4 x85 + 4 x89 + 2 x9f +
      8 x108 + 8 x1011 + 7 x119;

// fluxo que entra no vértice = fluxo que sai
vertice_i: xi0 + xi6 = 1;
vertice_0: xi0 = x01 + x04;
vertice_1: x01 = x12;
vertice_2: x12 + x42 = x23;
vertice_3: x23 + x53 = x3f;
vertice_4: x04 + x74 = x42 + x45;
vertice_5: x45 + x85 = x53 + x5f;
vertice_6: xi6 = x67 + x610;
vertice_7: x67 = x74 + x78;
vertice_8: x78 + x108 = x85 + x89;
vertice_9: x89 + x119 = x9f;
vertice_10: x610 = x108 + x1011;
vertice_11: x1011 = x119;
```

As variáveis de decisão que tomam o valor 1 na solução óptima definem o caminho mais longo entre os vértices i e f.

Minimizaçao do tempo de conclusão

Existe outro modelo de programação matemática para este problema em que cada variável de decisão $t_i, \forall i$, representa o tempo de início da actividade i, e em que o objectivo é minimizar o tempo de execução total do projecto obedecendo a todas as precedências.

As restrições do problema, relativas a cada um dos arcos do grafo, traduzem as relações de precedência entre as actividades. Para uma dada actividade j, o tempo de início da actividade j deve ser posterior ao tempo de conclusão de cada uma das actividades i que precedem j. Dado que t_i

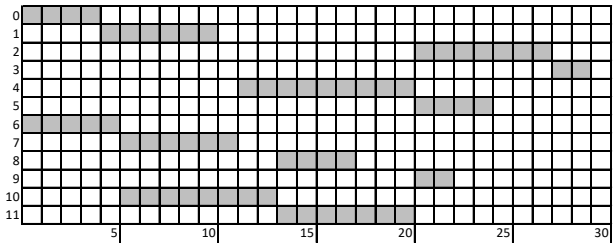


Figura 5.1: Diagrama de Gantt do projecto

designa o tempo de início da actividade i, a função $t_i + d_i$ designa o tempo de conclusão da actividade i. O projecto termina no instante de tempo t_f, quando todas as actividades predecessoras imediatas da actividade fictícia `fim` estiverem concluídas.

```
/* função objectivo */
min: tf ;

/* restrições */
arco_01: t1 >= t0 + 4 ;
arco_12: t2 >= t1 + 6 ;
arco_23: t3 >= t2 + 7 ;
arco_i0: t0 >= ti + 0 ;
arco_04: t4 >= t0 + 4 ;
arco_42: t2 >= t4 + 9 ;
arco_53: t3 >= t5 + 4 ;
arco_3f: tf >= t3 + 2 ;
arco_45: t5 >= t4 + 9 ;
arco_5f: tf >= t5 + 4 ;
arco_i6: t6 >= ti + 0 ;
arco_74: t4 >= t7 + 6 ;
arco_85: t5 >= t8 + 4 ;
arco_9f: tf >= t9 + 2 ;
arco_67: t7 >= t6 + 5 ;
arco_78: t8 >= t7 + 6 ;
arco_89: t9 >= t8 + 4 ;
arco_610: t10 >= t6 + 5 ;
arco_108: t8 >= t10 + 8 ;
arco_119: t9 >= t11 + 7 ;
arco_1011: t11 >= t10 + 8;
```

O valor da variável de decisão t_i na solução óptima define o tempo de início de execução da actividade i, permitindo construir um plano de execução do projecto, designado por diagrama de Gantt. O projecto é executado num tempo com a duração do valor da solução óptima. O Diagrama de Gantt do projecto em análise é apresentado na Figura 5.1.

5.3 Gestão de projectos: crashing times

Em gestão de projectos, há situações em que pode haver interesse em reduzir a execução do tempo de execução do projecto. Embora haja uma duração definida para cada actividade, é muitas vezes possível, aumentando os recursos nela aplicados, reduzir a sua duração. Isto é feito com custos suplementares, mas a reduzir da duração de execução do projecto global pode trazer benefícios.

A forma de modelar o aumento de custo em função da redução da duração da actividade pode ser feita de várias formas diferentes. Vamos assumir que as reduções de custo são lineares com o tempo, e associar a cada actividade três parâmetros adicionais: o primeiro é o valor do custo normal, expresso em unidades monetárias [U.M.], o segundo é o valor do custo suplementar de reduzir a duração da actividade de uma unidade de tempo [U.T.], expresso em [U.M./U.T.], e o terceiro é o valor da máxima redução de tempo que é permitida para a actividade, expresso em [U.T.]. Vamos considerar o exemplo apresentado na Secção anterior. Esses valores estão apresentados na seguinte Tabela:

Actividade	Custo Normal	Custo suplementar de redução	Máxima redução
0	400	100	1
1	1000	300	2
2	1400	500	4
3	300	100	1
4	2000	400	3
5	1000	800	1
6	800	90	2
7	900	–	0
8	600	100	1
9	300	—	0
10	1600	500	1
11	1400	300	2

A título ilustrativo, para a Actividade 1, cuja duração normal é de 6 U.T. e custo normal é 1000 U.M., reduzir a duração da actividade de 1 U.T., passando a 5 U.T., tem um custo suplementar de 300 U.M., ou seja, a Actividade 1 passa a ter um custo total de 1300 U.M.. A redução máxima que se pode obter é de 2 U.T., ou seja, é possível, com um custo suplementar de 600 U.M., realizar esta actividade no tempo mínimo de 4 U.T., com um custo total de 1600 U.M..

Suponha que se pretende que o tempo de execução do projecto seja reduzido em 3 U.T.. O objectivo do problema é decidir como devem ser reduzidas as durações das actividades, de modo a realizar o projecto na nova duração desejada, com um custo suplementar mínimo.

```
// custo associado à redução das durações das actividades
min: 100 r0 + 300 r1 + 500 r2 + 100 r3 + 400 r4 + 800 r5 +  90 r6 +   0 r7 +
      100 r8 +   0 r9 + 500 r10+ 300 r11;

// tempo máximo para concluir o projecto
tf <= 26;

// relações de precedência
// na restrição tj >= ti - ri + di, a função ti - ri + di designa
// o tempo de conclusão da actividade i após a redução da duração,
```

```
// de di para -ri + di

arco_01: t1 >= t0 - r0 + 4 ;
arco_12: t2 >= t1 - r1 + 6 ;
arco_23: t3 >= t2 - r2 + 7 ;
arco_i0: t0 >= ti + 0 ;
arco_04: t4 >= t0 - r0 + 4 ;
arco_42: t2 >= t4 - r4 + 9 ;
arco_53: t3 >= t5 - r5 + 4 ;
arco_3f: tf >= t3 - r3 + 2 ;
arco_45: t5 >= t4 - r4 + 9 ;
arco_5f: tf >= t5 - r5 + 4 ;
arco_i6: t6 >= ti + 0 ;
arco_74: t4 >= t7 - r7 + 6 ;
arco_85: t5 >= t8 - r8 + 4 ;
arco_9f: tf >= t9 - r9 + 2 ;
arco_67: t7 >= t6 - r6 + 5 ;
arco_78: t8 >= t7 - r7 + 6 ;
arco_89: t9 >= t8 - r8 + 4 ;
arco_610: t10 >= t6 - r6 + 5 ;
arco_108: t8 >= t10 - r10 + 8 ;
arco_119: t9 >= t11 - r11 + 7 ;
arco_1011: t11 >= t10 - r10 + 8;

// reduções máximas permitidas
r0 <= 1 ;
r1 <= 2 ;
r2 <= 4 ;
r3 <= 1 ;
r4 <= 3 ;
r5 <= 1 ;
r6 <= 2 ;
r7 <= 0 ;
r8 <= 1 ;
r9 <= 0 ;
r10 <= 1 ;
r11 <= 2 ;
```

Balanceamento entre a duração do projecto e seu custo

Há situações em que pode haver interesse em analisar o balanceamento entre a duração global do projecto e o seu custo de execução. Isso pode acontecer, por exemplo, em obras de construção de auto-estradas em que o contrato preveja que o construtor pode beneficiar do prémio de rceber os valores cobrados em portagem se a data de abertura da auto-estrada for antecipada em relação à data de entrega contratada. O uso do modelo apresentado serve fazer esse estudo, no

sentido de determinar a melhor opção. Tomando a duração global do projecto t_f como parâmetro, e resolvendo o modelo para cada valor do parâmetro, podemos obter o respectivo valor de custo suplementar, como se mostra na seguinte tabela:

tempo de execução	custo suplementar
29	0
28	90
27	180
26	280
25	680
24	1080
23	1480
22	1980
21	2480
20	2980
19	4580

É de salientar que não é possível executar o projecto numa duração de 18 U.T., por as reduções de duração máximas para as actividades não o permitirem.

Os resultados desta análise são muitas vezes apresentados num gráfico em que, no eixo das abscissas, se representam as diferentes durações possíveis para o projecto e, no eixo das ordenadas, se representam os custos totais (ou suplementares) de execução do projecto.

5.4 Substituição de equipamento

A solução do problema do caminho mais curto permite determinar a política óptima de substituição de equipamento. No seguinte exemplo, pretende-se minimizar os custos de operação de um dado equipamento cujos custos de manutenção crescem com o tempo.

Uma companhia necessita de uma máquina de um determinado tipo para os próximos n anos. Neste horizonte de planeamento, a companhia pode, no início de qualquer ano, proceder à substituição da máquina existente, evitando assim incorrer custos de manutenção crescentes. O custo de manutenção para uma máquina com k anos é conhecido, bem como o seu valor residual.

Seja c_{ij} o custo total de manter uma máquina em funcionamento desde o início do período i até ao início do período j, compreendendo o custo de aquisição, os custos de manutenção no fim de cada ano e a recuperação do valor residual, no fim da vida útil da máquina. Este problema pode ser modelado e representado através de um grafo acíclico. Para um horizonte de planeamento de n anos, considera–se um conjunto de $n+1$ vértices, cada um representando o início de um ano.

Para um horizonte de 4 anos, obtém–se o grafo apresentado na Figura 5.2.

Um caminho desde o vértice 1 até ao vértice 5 garante a existência de uma máquina em funcionamento durante os quatro anos. Por exemplo, o caminho definido pelos arcos (1,4) e (4,5) corresponde a adquirir uma máquina no início do primeiro ano e a vendê–la no fim do terceiro ano de funcionamento, altura em que é adquirida uma máquina nova, que, por sua vez, é mantida durante um ano.

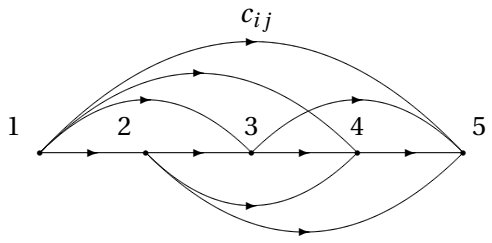


Figura 5.2: Modelo de substituição de equipamento

5.5 Escalonamento de equipas para cobrir tarefas com datas fixas

O objectivo deste problema é determinar o número mínimo de equipas que é necessário escalonar para a realização de um determinado conjunto de tarefas com horários pré-definidos. Este tipo de problemas é classificado como sendo de escalonamento fixo, ou *fixed schedule*, na literatura anglo-saxónica. O problema de maximização de fluxo pode ser usado na solução de problemas deste tipo.

Vamos usar um exemplo em que se pretende fazer o escalonamento de viaturas para cobrir um conjunto de tarefas com datas fixas, e há um tempo de deslocação entre os pontos onde cada tarefa é realizada.

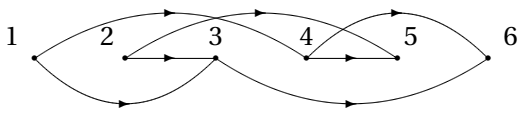
Considere um conjunto de tarefas, designadas pelo índice $i, i = 1,2,\ldots,n$. Cada tarefa i tem associado um tempo de início, t_i, uma duração, d_i, um local de início, I_i, e um local de fim, F_i. Após a realização de um serviço i, a viatura pode deslocar–se para outro local, para aí dar início ao serviço j; esta deslocação, que corresponde a um tempo morto, tem uma duração bem definida d_{ij}. Assim, a tarefa i pode preceder a tarefa j se $t_j \geq t_i + d_i + d_{ij}$.

O objectivo único do problema é minimizar o número de viaturas necessárias. Todos os outros custos são desprezados. Considera–se que o custo de atribuir uma viatura extra é sempre superior a quaisquer outros custos, nomeadamente os custos de deslocação entre os vários locais.

O problema pode ser representado por um grafo orientado, com vértices $1,2,\ldots,n$, cada um correspondendo a uma tarefa, e arcos (i,j), se $t_j \geq t_i + d_i + d_{ij}$, indicando que a tarefa j pode ser executada depois da tarefa i. O grafo assim construído é acíclico.

Cada caminho corresponde a uma sequência de serviços efectuados por uma viatura. Esses serviços são os vértices que pertencem ao caminho.

Exemplo 5.1. Considere o conjunto de 6 tarefas para as quais foram identificadas as precedências indicadas na Figura. Os arcos $(2,3)$ e $(2,5)$ significam que, depois de executar a tarefa 2, uma viatura pode executar, quer a tarefa 3, quer a tarefa 5, não sendo viável executar nenhuma outra tarefa.



A cada caminho, corresponde uma sequência de vértices (tarefas) que podem ser realizados por uma viatura. A título ilustrativo, uma viatura pode executar as tarefas 2, 3 e 6. □

A determinação do número mínimo de viaturas necessárias para realizar todos os serviços pode ser feita decompondo o grafo num conjunto mínimo de caminhos. Este problema da decomposição mínima em caminhos de um grafo acíclico pode ser resolvido através de um problema de maximização de fluxo, definido num grafo auxiliar. Este facto deriva do seguinte teorema:

Teorema 3. *Considere um grafo acíclico G com n vértices. Seja c o número de caminhos resultantes de uma decomposição que cobre todos os vértices de G. Sendo a o número total de arcos desses caminhos,*

$$a + c = n.$$

Prova: Cada caminho é constituído por a_j arcos, $j = 1, 2, \ldots, c$, e por $a_j + 1$ vértices. Em particular, se uma viatura executar apenas uma tarefa, existe um caminho apenas com um vértice e sem arcos. Cada vértice irá fazer parte de um e de apenas um caminho, porque cada tarefa deve ser executada apenas por uma viatura. Como todas as tarefas devem ser executadas, a soma dos vértices de todos os caminhos deve perfazer o número total de vértices:

$$n = \sum_{j=1}^{c} (a_j + 1) \quad = \quad \sum_{j=1}^{c} a_j + c = a + c$$

□

Sendo a soma das duas quantidades constante, minimizar o número de viaturas corresponde a maximizar o número de arcos seleccionados para integrar caminhos, problema que é resolvido maximizando o fluxo num grafo auxiliar G'.

Dado um grafo orientado acíclico G, o grafo auxiliar $G' = (V_1, V_2, A')$ é um grafo bipartido com $2n$ vértices, sendo cada vértice de G dividido em dois vértices j e j', pertencentes a V_1 e V_2, respectivamente; por cada arco (i, j) de G, haverá um arco (i, j') em G'. Adicionalmente, G' terá uma fonte s, com arcos para todos os vértices de V_1, e todos os vértices de V_2 são ligados a um terminal t. Sendo necessário garantir que cada arco pertence apenas a um caminho, que corresponde a que cada tarefa seja apenas executada por uma viatura, a capacidade de todos os arcos de G' é unitária.

A solução que maximiza o fluxo fornece imediatamente a decomposição em caminhos pretendida. Os arcos que fazem parte da solução são os arcos que integram os caminhos.

Exemplo 5.2 (cont.)**.** O grafo G' é apresentado na Figura 5.3. A solução óptima tem um fluxo máximo de 4 unidades, com fluxos unitários entre os vértices $(1, 4')$, $(2, 3')$, $(3, 6')$ e $(4, 5')$. Esta solução corresponde à utilização de 2 viaturas. A primeira executa as tarefas 1, 4 e 5, enquanto a segunda executa as tarefas 2, 3 e 6.

□

5.6 Planeamento com janelas temporais

O problema de planeamento sujeito a restrições temporais em que é possível interromper a execução das tarefas pode ser modelado como um problema de fluxo máximo. Pretende-se determinar como se deve, e se é possível, realizar um conjunto de trabalhos com os recursos humanos existentes num dado período de planeamento utilizando os recursos disponíveis.

Os trabalhos são caracterizados por uma duração, por uma data de lançamento, data a partir da qual podem ser realizados, e um prazo de entrega, e por uma janela temporal, que é o intervalo de tempo dentro do qual a tarefa pode ser executada.

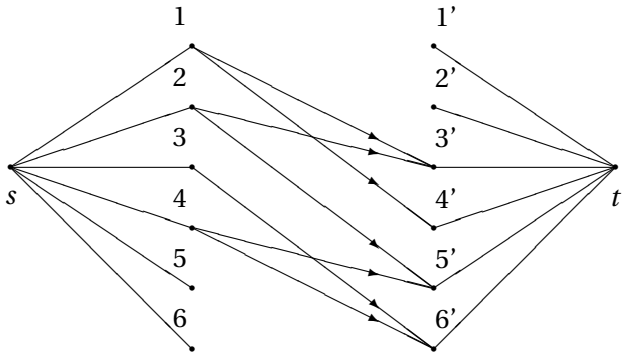


Figura 5.3: Grafo Auxiliar G'

Este modelo apenas se aplica em situações em que é permitido realizar a tarefa com interrupções, ou seja, não é necessário que a tarefa seja realizada continuamente desde o início até ao fim, podendo ser abandonada e retomada mais tarde.

Este problema pode ser formulado como um problema de maximização de fluxo definido sobre um grafo bipartido. De um lado, existe um conjunto de nós representando os períodos e, do outro lado, um conjunto de nós representando as tarefas. Existirá um arco entre um período e uma tarefa se for possível realizar tarefa nesse período. A capacidade do arco depende do número de pessoas que podem estar a realizar simultaneamente a tarefa. Se apenas 1 pessoa puder estar atribuída a cada trabalho, a capacidade é unitária.

Para completar a formulação, considere dois nós adicionais com ligações aos nós já referidos. Um nó-fonte, com arcos dirigidos para cada período. A capacidade destes arcos é igual ao número de pessoas disponíveis, ou seja, ao número de tarefas que podem estar a ser realizadas simultaneamente. Ainda, um nó-terminal, com arcos provenientes de cada tarefa. A capacidade destes arcos é igual ao número de períodos × pessoas necessárias à conclusão da tarefa.

A solução do problema permite determinar se a obra é exequível com as condições impostas. Se existir um fluxo que sature os arcos entre as tarefas e o nó-terminal, então é possível assegurar a realização de todas as tarefas. Saturar estes arcos significa contribuir para a tarefa com o número requerido de períodos×pessoaa. Caso contrário, não é possível realizar o conjunto de tarefas com os recursos disponíveis.

Exemplo 5.3. Considere o quadro da Figura 5.4 que apresenta um conjunto de 5 tarefas com as durações indicadas e os respectivos limites temporais representados por barras. O período de planeamento é de 5 semanas e existem 2 pessoas disponíveis. Pretende-se determinar como se deve, e se é possível, realizar o conjunto de tarefas com os recursos humanos existentes. O modelo de maximização de fluxo correspondente é apresentado no grafo da Figura.

5.7 Fecho máximo de um grafo (*maximum closure*)

Dado um grafo orientado $G = (V, A)$, em que cada vértice $j \in V$ tem associado um peso, c_j, que pode ser positivo ou negativo, e arcos $(i, j) \in A$ (orientados), o problema do fecho máximo de um grafo, conhecido na literatura anglo-saxónica como *maximum closure*, consiste em determinar o subconjunto fechado de vértices $S \subseteq V$, tal que $\sum_{j \in S} c_j$ seja máximo. Um subconjunto S é *fechado* se todos os sucessores dos vértices de S pertencerem também a S.

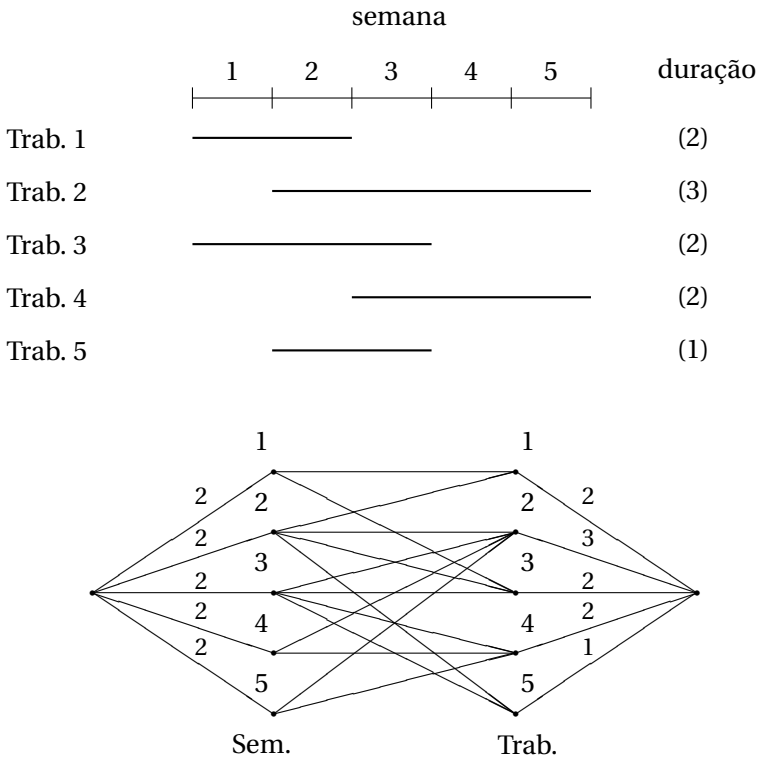


Figura 5.4: Modelo de Planeamento com Janelas Temporais

O problema de fecho máximo de um grafo pode ser formulado como um problema de programação linear com variáveis binárias. Seja x_j uma variável binária que toma o valor 1 se o vértice j pertencer ao fecho do grafo, e 0, caso contrário. A formulação matemática do problema do fecho máximo de um grafo é a seguinte:

$$\max \quad \sum_{j \in V} c_j x_j \tag{5.1}$$

$$\text{suj. a} \quad x_i - x_j \leq 0, \forall (i,j) \in A \tag{5.2}$$

$$x_j \text{ binário}, \forall\, j \in V \tag{5.3}$$

As restrições (5.2) são restrições lógicas de implicação, que traduzem que, se o vértice i pertencer ao fecho de um grafo, então o vértice j também deve pertencer.

O fecho máximo de um grafo pode ser determinado resolvendo um problema de fluxo máximo num grafo auxiliar $G' = (V', A')$, em que o conjunto de vértices de G' tem dois vértices adicionais, uma fonte s e um terminal t, *i.e.*, $V' = V \cup \{s,t\}$. Seja $V^+ = \{j \in V : c_j \geq 0\}$ e $V^- = \{j \in V : c_j < 0\}$. O grafo auxiliar G' terá arcos (s,j) de s para todos os vértices $j \in V^+$, com custo c_j, e arcos (j,t) de todos os vértices $j \in V^-$ para t, com custo $-c_j$. Notar que este valor é positivo, dado os vértices de V^- serem os vértices com peso $c_j < 0$. Os arcos pertencentes a G devem ter uma capacidade igual a $+\infty$ no grafo G'. O fecho máximo do grafo G é determinado pelo corte mínimo (s,t) do grafo G', o problema dual do problema de fluxo máximo.

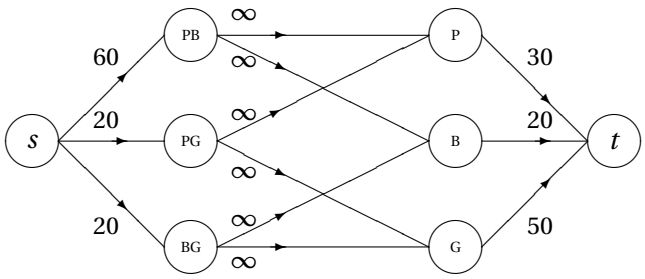
Exemplo 5.4. Uma empresa de camionagem está a estudar a possibilidade de estabelecer novas carreiras numa área geográfica onde não opera actualmente. Os lucros esperados associados a cada carreira são os apresentados na seguinte Tabela:

Carreira	Lucro (U.M.)
Porto-Braga	60
Porto-Guimarães	20
Braga-Guimarães	20

Para operar a partir de uma dada cidade, a companhia precisa de instalações para venda de bilhetes e armazenamento de carga. O custo de cada instalação é fixo, independentemente do número de carreiras com base nessa cidade. Os seus valores são dados pela seguinte Tabela:

Instalação	Custo (U.M.)
Porto	30
Braga	20
Guimarães	50

O objectivo deste problema é seleccionar o conjunto de carreiras que maximiza o lucro de operação, definido como a diferença entre o lucro das carreiras seleccionadas e o custo das instalações. Considere um modelo em que existe um nó para cada carreira, e um arco desde o vértice s para esse nó com uma capacidade igual ao lucro associado à carreira. Por outro lado, existem arcos que ligam os nós associados às cidades a um vértice t com uma capacidade igual ao custo da instalação respectiva. Entre os nós correspondentes a uma dada carreira e as cidades entre as quais opera, existe um arco de capacidade infinita.



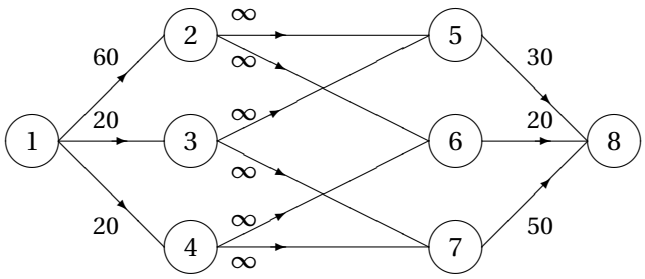
A cada selecção válida, corresponde um corte s,t de valor finito em que, num dos conjuntos da partição, estão o nó s e os nós das carreiras seleccionadas e das cidades por elas usadas. Nenhum dos arcos de capacidade infinita pode fazer parte de um corte mínimo, pelo que a selecção de uma carreira implica a selecção da cidade por ela usada. Isto significa que os nós das carreiras e cidades usadas vão pertencer ao mesmo conjunto da partição. Os arcos que atravessam o corte definem a sua capacidade. Neste modelo, o valor da capacidade do corte é igual ao lucro das carreiras não seleccionadas mais os custos associados às instalações usadas.

Seja S o conjunto das carreiras seleccionadas, $\bar{S}$ o seu complemento e I o conjunto de instalações usadas. Vamos designar por l_j o lucro da carreira j e por c_i o custo da instalação i.

$$\begin{aligned}
\text{lucro de operação} &= \sum_{j\in S} l_j - \sum_{i\in I} c_i \\
&= \sum_{j\in S} l_j + \sum_{j\in \bar{S}} l_j - (\sum_{j\in \bar{S}} l_j + \sum_{i\in I} c_i) \\
&= \sum_{j\in (S\cup \bar{S})} l_j - \text{capacidade do corte}
\end{aligned}$$

O lucro da operação é dado pela diferença entre o valor da soma do lucro de todas as carreiras e o valor da capacidade do corte. Como a primeira parcela é uma constante, minimizar a capacidade do corte equivale a maximizar o lucro da operação.

Para construir o ficheiro de *input* do *Relax4*, vamos usar a numeração de vértices indicada na Figura seguinte. O fluxo máximo no grafo é dado pelo fluxo no arco $(8,1)$, um arco entre o terminal e a fonte, por onde se faz o retorno do fluxo que atravessa o grafo, dando origem ao que se designa por circulação de fluxo. Este arco $(8,1)$ não está representado na Figura. Notar que todos os valores das ofertas / consumos dos vértices são nulos. O objectivo é maximizar o fluxo, e portanto devemos associar ao arco de retorno um custo unitário de transporte igual a -1, dado que o *Relax4* assume que todos os problemas são de minimização. Assim, deve-se minimizar a função simétrica.



O ficheiro de input do *Relax4* é:

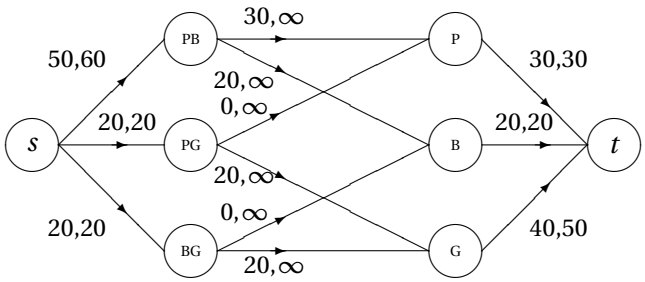
```
8
13
1 2  0    60
1 3  0    20
1 4  0    20
2 5  0  1000
2 6  0  1000
3 5  0  1000
3 7  0  1000
4 6  0  1000
4 7  0  1000
5 8  0    30
6 8  0    20
7 8  0    50
8 1 -1  1000
0
0
0
0
0
0
0
0
```

A solução óptima do problema é a seguinte.

```
C:\Users\vc\Desktop\Dossier IO\RELAX4 2013>relax4 <FechoMaximo.txt
 END OF READING
 NUMBER OF NODES = 8, NUMBER OF ARCS = 13
 CONSTRUCT LINKED LISTS FOR THE PROBLEM
 CALLING RELAX4 TO SOLVE THE PROBLEM
 **********************************
 TOTAL SOLUTION TIME =  0. SECS.
 TIME IN INITIALIZATION =  0. SECS.
   1 2  50.
```

```
  1 3  20.
  1 4  20.
  2 5  30.
  2 6  20.
  3 7  20.
  4 7  20.
  5 8  30.
  6 8  20.
  7 8  40.
  8 1  90.
OPTIMAL COST =  -90.
NUMBER OF AUCTION/SHORTEST PATH ITERATIONS = 12
NUMBER OF ITERATIONS =  28
NUMBER OF MULTINODE ITERATIONS =  5
NUMBER OF MULTINODE ASCENT STEPS =  0
NUMBER OF REGULAR AUGMENTATIONS =  5
*********************************
```

Os valores do fluxo na solução óptima são apresentados na Figura.



O corte mínimo do exemplo apresentado tem o valor de 90, correspondendo à selecção da carreira PB e a instalações no Porto e em Braga. O lucro máximo de operação é 10. □

5.8 Implantação de máquinas num *layout* existente

Os problemas de concepção de *layouts* visa determinar os melhores locais para colocar equipamentos no chão de fábrica, de modo a minimizar os tempos necessários à transferência de peças que são processadas pelas máquinas. Há muitas variantes de problemas de *layouts*. O problema a seguir apresentado surge numa situação em que se pretende colocar novas máquinas num *layout* existente. Esta variante pode ser resolvida como um problema de afectação.

Três máquinas novas, designadas por R, S e T, devem ser implantadas num *layout* já existente, constituído por 5 máquinas. O número estimado de transferências diárias entre as máquinas existentes e as máquinas novas é dado pelo seguinte Quadro:

	Máquinas existentes				
Máq. Nova	A	B	C	D	E
R	0	7	0	2	7
S	3	0	15	0	2
T	10	5	0	3	0

Não estão previstos movimentos entre as máquinas novas. Existem 4 possíveis locais onde qualquer uma das 3 máquinas pode ser implantada. As distâncias entre esses locais e as máquinas existentes são dados pelo seguinte Quadro:

	Máquinas existentes				
Local Possivel	A	B	C	D	E
W	1	3	7	4	5
X	7	10	7	8	1
Y	8	4	7	5	4
Z	10	6	1	10	6

Todos os transportes são efectuados por uma ponte rolante, sendo os custos de transporte proporcionais à distância percorrida.

Para determinar a melhor solução de implantação, é necessário avaliar os custos de manuseamento resultantes da implantação de cada máquina em cada local. Cada máquina pode ser implantada num dos 4 locais disponíveis para o efeito. Dependendo da escolha, serão incorridos custos de transporte que dependem não apenas das distâncias, mas também da quantidade de movimentos entre o local escolhido e as máquinas existentes. A título ilustrativo, se a máquina R for implantada no local W, teremos um custo de manuseamento dado pela seguinte expressão:

$$1 \times 0 + 3 \times 7 + 7 \times 0 + 4 \times 2 + 5 \times 7 = 64$$

O cálculo dos valores de todas as possibilidades produz a seguinte matriz:

	Locais possíveis			
Máq. Nova	W	X	Y	Z
R	64	93	66	104
S	118	128	137	57
T	37	144	115	160

A solução que minimiza os custos totais de manuseamento de material define os locais onde as máquinas devem ser implantadas. O ficheiro de input do LPSolve é o seguinte:

```
/*
      Implantação de  Máquinas num layout existente
      Problema de afectação
*/
min:  64 x11 + 93 x12 + 66 x13 +104 x14
     +118 x21 +128 x22 +137 x23 + 57 x24
     + 37 x31 +144 x32 +115 x33 +160 x34
     +  0 x41 +  0 x42 +  0 x43 +  0 x44;

/*     A máquina deve ser implantada num local        */
```

```
        x11 +   x12 +   x13 +   x14   =  1 ; // màquina R
        x21 +   x22 +   x23 +   x24   =  1 ; // máquina S
        x31 +   x32 +   x33 +   x34   =  1 ; // máquina T
        x41 +   x42 +   x43 +   x44   =  1 ; // máquina fictícia
//      A restrição da máquina fictícia modela o facto de um dos
//      locais possíveis de implantação ficar vazio

/*      Cada local pode receber apenas uma máquina    */
        x11 +   x21 +   x31 +   x41   =  1 ;  // local W
        x12 +   x22 +   x32 +   x42   =  1 ;  // local X
        x13 +   x23 +   x33 +   x43   =  1 ;  // local Y
        x14 +   x24 +   x34 +   x44   =  1 ;  // local Z
```

A solução óptima do problema de afectação dada pelo LPSolve tem um custo óptimo de 160, correspondendo à solução $x_{13} = x_{24} = x_{31} = x_{42} = 1$, e as restantes variáveis iguais a 0. Essa solução corresponde a implantar a máquina R no local Y, a máquina S no local Z e a máquina T no local W, ficando o local X vazio.

5.9 Escala de rotação de veículos

Nos problemas de escala de rotação de veículos pretende-se determinar o número mínimo de veículos que são necessários para assegurar o transporte de mercadorias em várias rotas cumpridas regularmente. A determinação da escala de rotação de veículos pode ser resolvida como um problema de afectação.

Considere o Exemplo de um operador logístico que efectua diariamente as seguintes rotas entre diversas cidades europeias:

rota	origem	destino
BE	Barcelona	Estugarda
GC	Genebra	Calais
FH	Florença	Hamburgo
DE	Dresden	Estugarda
AG	Amesterdão	Genebra
CA	Calais	Amesterdão

As distâncias entre estas cidades, medidas em dias de viagem, são as indicadas no seguinte Quadro:

	B	C	D	E	F	G	H
A	4	1	2	2	3	2	1
B		3	5	3	3	2	5
C			3	2	3	2	2
D				1	3	3	1
E					2	1	2
F						2	4
G							3

As rotas acima referidas efectuam-se apenas no sentido indicado. Se, para cada rota, fosse também necessário assegurar a rota no sentido contrário, o problema de determinar o número mínimo de camiões seria simples. A melhor solução, para cada veículo, seria efectuar as duas rotas, de ida e de volta, viajando sempre entre as mesmas cidades. Tal não acontece na generalidade dos casos. Assim, é necessário decidir, qual a sequência de rotas que cada veículo deve assegurar.

Este problema pode ser formulado como um problema de minimização dos tempos das deslocações entre o fim de cada rota e o início da seguinte. Minimizar os tempos perdidos em deslocações acaba por traduzir-se em minimizar a dimensão da frota.

A solução óptima do problema de determinar a frota mínima corresponde à solução dada por um problema de afectação, cujos custos c_{ij} são definidos como o número de dias que decorrem desde o dia do fim da rota i até ao dia do início da rota j, ou seja, o tempo de deslocação para, depois de terminar uma rota, o camião se posicionar para iniciar a rota seguinte:

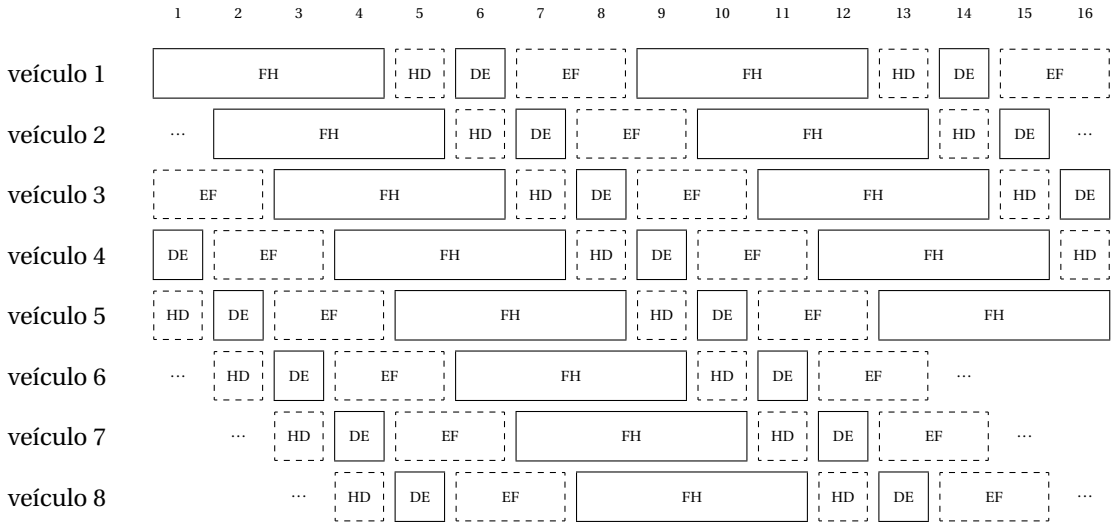
	BE	GC	FH	DE	AG	CA
BE	3	[1]	2	1	2	2
GC	3	2	3	3	1	[0]
FH	5	3	4	[1]	1	2
DE	3	1	[2]	1	2	2
AG	[2]	0	2	3	2	2
CA	4	2	3	2	[0]	1

A título de exemplo, transitar da rota entre Barcelona e Estugarda para a mesma rota, o que implica fazer a viagem de regresso entre Estugarda e Barcelona sem carga, traduz-se numa perda de 3 dias. É de salientar que a matriz não é simétrica.

A solução óptima do problema de afectação tem um valor de 6. Há várias soluções óptimas alternativas, mas vamos usar a solução indicada no quadro para estabelecer uma escala de rotação de veículos. A solução corresponde a afectar um conjunto de veículos às rotas BE–GC–CA–AG–BE, e um conjunto às rotas FH–DE–FH.

O valor da solução do problema de afectação traduz o número de veículos que são necessário, em cada dia, para fazer as deslocações entre rotas. Além destes, são precisos mais 13 veículos (a soma dos tempos das 6 rotas), em cada dia, para o transporte efectivo de mercadorias, o que perfaz um total de 19 veículos para assegurar o cumprimento diário de todas as rotas.

Há 11 veículos que ciclicamente fazem as rotas BE–GC–CA–AG–BE, enquanto 8 veículos fazem as rotas FH–DE–FH. Para estes últimos, o mapa de rotação de veículos para um período de 16 dias é o indicado no seguinte Quadro, em que os rectângulos a cheio representam o transporte numa rota, enquanto os rectângulos a tracejado correspondem a uma deslocação entre o destino de um transporte e a origem do seguinte:



No caso de haver necessidade de utilizar mais do que um veículo diariamente em cada rota, o problema poderia ser formulado como um problema de transportes.

5.10 Escalonamento de veículos de um depósito

O objectivo é minimizar o custo total da operação de uma frota de veículos que têm de visitar um conjunto de clientes. Na função custo, são incluídos os custos de deslocação e os custos fixos de utilização de veículos. A solução consiste em decidir quais os itinerários a seguir por cada veículo de modo a minimizar o custo total. O problema que vamos analisar é uma extensão do problema de escalonamento de equipas para cobrir tarefas com datas fixas, apresentado na Secção **??**, em que também existe uma data de serviço associada ao serviço de um cliente, *i.e.*, é um problema de *fixed scheduling*. Este problema pode ser resolvido como um problema de afectação. Nota-se que o caso geral é um problema difícil que envolve a determinação dos clientes que são servidos por cada veículo e da ordem pela qual devem ser visitados.

Portanto, nesta versão, a cada serviço é associado um instante de execução a_j. O serviço do cliente j pode ser executado no instante a_j, se, após terminar um serviço no cliente i, o veículo puder chegar ao cliente j antes do instante a_j. É permitido chegar antes, mas é necessário esperar pelo instante de execução do serviço. Vamos considerar que a duração do serviço no cliente é desprezável, com valor nulo, notando que se a duração fosse positiva, essa condição não é difícil de incluir no modelo. Os custos de deslocação entre cidades, c_{ij}, incluindo despesas de combustível, portagens, e outros, são os indicados no seguinte Quadro:

	B	C	D	E	F	G	H
A	13	5	6	5	10	7	1
B		11	14	10	8	6	15
C			8	6	10	6	2
D				4	8	8	4
E					6	4	6
F						5	11
G							9

A carga não constitui restrição, podendo uma viagem visitar um qualquer número de clientes. As datas de serviço associadas aos clientes são as indicadas no seguinte Quadro:

destino	data do serviço
Amesterdão	2
Barcelona	7
Calais	4
Dresden	2
Estugarda	10
Florença	6
Genebra	9

Dado um conjunto de datas de serviço, é possível estabelecer quais os clientes j que é possível visitar depois de ter visitado o cliente i. Essa informação pode ser representada num grafo em que existe um arco entre o vértice i e o vértice j, se tal sequência for possível. Dado que existe um tempo associado a cada serviço, o grafo é acíclico. O grafo pode ser completado adicionando um vértice correspondendo ao depósito, que deve funcionar como início e fim da viagem de cada veículo.

O itinerário de cada veículo é definido pela solução de um problema de afectação. As variáveis x_{ij} com valor 1 determinam que o veículo deve efectuar o trajecto entre o cliente i e o cliente j. Ao problema de afectação, é adicionada uma linha e uma coluna extra por cada veículo disponível no depósito. O itinerário de um veículo terá início no depósito, percorrerá um conjunto de clientes, e regressará ao depósito. Nunca poderá haver um itinerário que não passe pelo depósito, e que apenas percorra um conjunto de clientes em circuito fechado. Tal não acontece porque existem datas de serviço bem determinadas, e um veículo não pode regressar ao mesmo cliente.

Iremos considerar duas versões deste problema, uma com selecção do número de veículos e a outra com um número de veículos fixo.

Selecção do número de veículos

Na versão de selecção do número de veículos, é imposto um número máximo de veículos disponíveis, sendo o número de veículos efectivamente utilizados determinado pela solução óptima do problema. A função objectivo entra em linha de conta com os custos de deslocação e os custos fixos de utilização de cada veículo.

Vamos considerar um Exemplo em que existe uma frota de 3 veículos, e os custos de utilização fixos de 1 veículo são de 1 U.M.. O problema de afectação correspondente é o apresentado no seguinte Quadro:

	A	B	C	D	E	F	G	H1	H2	H3
A		13	5		5	10	7	1	[1]	1
B					10		[6]	15	15	15
C		[11]			6		6	2	2	2
D		14			4	8	8	[4]	4	4
E								6	6	[6]
F					[6]		5	11	11	11
G					4	[5]		9	9	9
H1	2	16	3	[5]	7	12	10	0	0	0
H2	[2]	16	3	5	7	12	10	0	0	0
H3	2	16	[3]	5	7	12	10	0	0	0

Todos os valores não representados na matriz correspondem a $+\infty$. Os custos finitos correspondem aos custos de deslocação das viagens permitidas, de acordo com as datas de serviço acima indicadas. A título ilustrativo, depois de visitar o cliente D, no dia 2, é possível fazer a deslocação para o cliente F, o que demora 3 dias, para servir o cliente F no dia 6. O custo associado a esta deslocação é de 8 U.M..

No Quadro, há várias linhas para o depósito H, tantas quantos os veículos disponíveis. Existem custos de deslocação de e para o depósito H. O custo fixo de utilização de um veículo é imputado ao arco de saída do depósito. Caso um veículo não seja utilizado (*e.g.*, $x_{H_1H_1} = 1$), esse custo não é incorrido.

A solução óptima deste Exemplo corresponde à utilização de 3 veículos e tem um valor de 49 U.M.. Os veículos fazem, respectivamente os trajectos, HDH, HAH e HCBGFEH.

Número de veículos fixo

Podemos impor a utilização de um número fixo de veículos, impondo custos muito elevados nas células entre depósitos. Neste caso, todos os veículos são forçados a sair do depósito. No Exemplo apresentado de seguida, são utilizados exactamente 2 veículos:

	A	B	C	D	E	F	G	H1	H2
A		13	[5]		5	10	7	1	1
B					10		[6]	15	15
C		[11]			6		6	2	2
D		14			4	8	8	[4]	4
E								6	[6]
F					[6]		5	11	11
G					4	[5]		9	9
H1	2	16	3	[5]	7	12	10		
H2	[2]	16	3	5	7	12	10		

A solução óptima tem um valor de 50 U.M.. Um dos veículos faz o trajecto HDH, enquanto o outro faz o trajecto HACBGFEH.

5.11 Recolha de materiais num armazém

Considere um armazém com a configuração indicada na Figura 5.5. Para proceder à manufactura de um produto, é necessário recolher os materiais que constam na ordem de fabrico. Essa tarefa é executada por um veículo programável de transporte, que procede à recolha automática dos objectos. O veículo programável pode deslocar-se ao longo de qualquer corredor, sendo o tempo de deslocação proporcional à distância percorrida.

O objectivo do problema consiste em determinar, para uma dada ordem de fabrico, o modo como deve ser programada a recolha dos objectos pelo veículo automático, de modo a minimizar o tempo total da operação.

Este problema pode ser formulado como um problema do caixeiro viajante, apresentado na Secção 3.10. Dadas as características particulares da matriz de custos, pode ser resolvido em tempo polinomial.

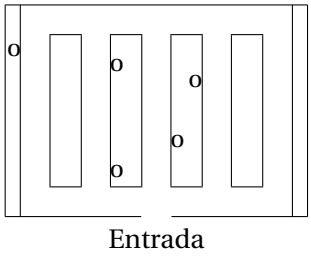


Figura 5.5: Configuração do armazém e posição dos objectos

5.12 Sequenciação de tarefas com custos dependentes da sequência

Há situações em que se pretende sequenciar um conjunto de n trabalhos numa única máquina em que os trabalhos podem ser executados por qualquer ordem, mas existem custos de preparação da máquina, que dependem do trabalho que foi efectuado anteriormente. A existência de custos de preparação dependentes da sequência pode assumir particular relevância.

Cada trabalho é caracterizado por uma grandeza física, como, por exemplo, a temperatura do forno ou a cor de um tanque de tingimento. Os custos de preparação podem estar relacionados com o ajuste da temperatura do forno para proceder à preparação de uma nova liga, que requer uma temperatura diferente da anterior. Este tipo de problemas pode ser formulado como um probema de caixeiro viajante, apresentado na Secção 3.10.

Considere o problema de sequenciação de um conjunto de n tarefas num forno industrial. Cada tarefa i é caracterizada pela temperatura inicial e final do forno, designadas por I_i e $F_i, i = 1,2,\ldots,n$. O custo de preparar o forno para executar a tarefa j depois de ter processado a tarefa i depende da temperatura final da tarefa i, F_i, e da temperatura inicial da tarefa j, I_j.

Há dois casos a considerar. Se a temperatura final da tarefa i for inferior ou igual à temperatura inicial da tarefa j é necessário aumentar a temperatura do forno, o que se traduz num custo de aquecimento. Por outro lado, se a temperatura final da tarefa i for superior à temperatura inicial da tarefa j é necessário baixar a temperatura do forno. Eventualmente, pode ser possível recuperar alguma da energia resultante do arrefecimento do forno, podendo haver lugar a um lucro associado a esta transição.

Vamos considerar que existe um custo de aquecimento do forno de 3 U.M./$10^3\ ^oC$ e que existe um lucro de recuperação de energia associado ao arrefecimento do forno de 1 U.M./$10^3\ ^oC$. As temperaturas iniciais e finais de cada uma das tarefas (em milhares de graus Celsius) são as indicadas no seguinte quadro:

tarefa	temp. inicial ($\times 10^3\ ^oC$)	temp. final ($\times 10^3\ ^oC$)
1	0.7	1.2
2	1.7	0.9
3	1.0	1.0
4	1.3	1.8
5	0.4	1.5

Os custos de preparação c_{ij} são dados pela seguinte matriz:

	1	2	3	4	5
1	–	1.5	-0.2	0.3	-0.8
2	-0.2	–	0.3	1.2	-0.5
3	-0.3	2.1	–	0.9	-0.6
4	-1.1	-0.1	-0.8	–	-1.4
5	-0.8	0.6	-0.5	-0.2	–

Se estas tarefas devessem ser realizadas ciclicamente, a solução óptima do problema é a sequência $1 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow 2 \rightarrow 1$, correspondendo a um custo total de preparação de -1.3, o que é equivalente a um lucro, que resulta da recuperação de energia nas transições entre tarefas. Globalmente, existe um custo positivo, porque, na execução das próprias tarefas, é necessário gastar energia sempre que a temperatura final da tarefa é superior à sua temperatura inicial. Claramente, esse gasto depende do processo tecnológico, e não podemos alterá-lo. Podemos apenas escolher a sequência de operações, e aí podemos obter economias.

O ficheiro de *input* do modelo é o seguinte:

```
/*
      Sequenciação de tarefas com custos dependentes da sequência
      Exemplo do forno
*/

min:             1.5 x12 -0.2 x13 +0.3 x14 -0.8 x15
      -0.2 x21            +0.3 x23 +1.2 x24 -0.5 x25
      -0.3 x31 +2.1 x32           +0.9 x34 -0.6 x35
      -1.1 x41 -0.1 x42 -0.8 x43           -1.4 x45
      -0.8 x51 +0.6 x53 -0.5 x53 -0.2 x54          ;

/*     Há apenas um arco a sair do vértice      */
                x12 +   x13 +   x14 +   x15 =  1 ; // vértice 1
       x21          +   x23 +   x24 +   x25 =  1 ; // vértice 2
       x31 +   x32          +   x34 +   x35 =  1 ; // vértice 3
       x41 +   x42 +   x43          +   x45 =  1 ; // vértice 4
       x51 +   x53 +   x53 +   x54          =  1 ; // vértice 5

/*     Há apenas um arco a entrar no vértice    */
                x21 +   x31 +   x41 +   x51 =  1 ;  // vértice 1
       x12          +   x32 +   x42 +   x52 =  1 ;  // vértice 2
       x13 +   x23          +   x43 +   x53 =  1 ;  // vértice 3
       x14 +   x24 +   x34          +   x54 =  1 ;  // vértice 4
       x15 +   x25 +   x35 +   x45          =  1 ;  // vértice 5
```

Resolvendo o modelo com o LPSolve, obtém-se o seguinte relatório com a solução óptima:

Variables	result
	-1,3
x12	0
x13	1
x14	0
x15	0
x21	1
x23	0
x24	0
x25	0
x31	0
x32	0
x34	0
x35	1
x41	0
x42	1
x43	0
x45	0
x51	0
x53	0
x54	1
x52	0

Salienta-se que a solução óptima do problema de caixeiro viajante é a solução óptima do problema de afectação. Caso a solução óptima do problema de afectação tivesse subcircuitos, seria necessário adicionar restrições para eliminar esses subcircuitos, e obter um circuito Hamiltoniano.

Dada a sua estrutura de custos, este problema constitui um caso especial do problema do caixeiro viajante que pode ser resolvido em tempo polinomial. Esta propriedade mantém-se válido para custos de preparar o forno para executar a tarefa j depois de ter processado a tarefa i dados por:

$$c_{ij} = \int_{F_i}^{I_j} f(x)dx \text{ , se } F_i \leq I_j$$

$$c_{ij} = \int_{I_j}^{F_i} g(x)dx \text{ , se } F_i > I_j,$$

sendo $f(x)$ e $g(x)$ duas funções quaisquer da temperatura x, tal que $f(x) + g(x) \geq 0$.

Existe uma variante deste problema, em que a realização das tarefas não é cíclica, mas apenas se pretende realizar cada tarefa uma vez. Essa variante também se pode resolver como um problema de caixeiro viajante, em que a matriz que representa os custos de preparação tem uma linha e uma coluna adicional, correspondente a um vértice fictício do grafo, que representa simultaneamente o início e o fim do plano de execução.

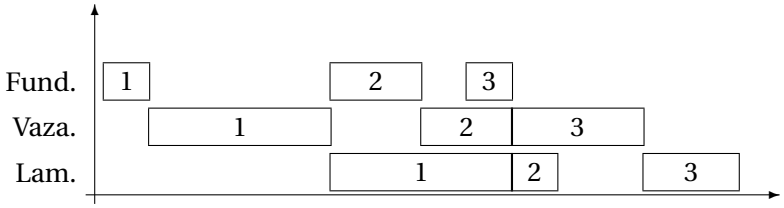
5.13 Flowshop sem inventários intermédios

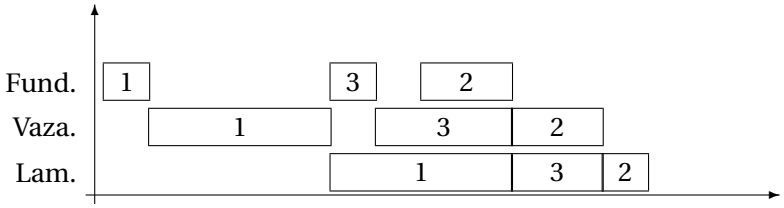
Existe uma classe de problemas de planeamento da produção em que as próprias características do processo obrigam a que não possa haver esperas entre as diversas operações que constituem uma tarefa, *i.e.*, logo que uma operação termine, a operação seguinte deve ser imediatamente iniciada. Estas situações ocorrem, por exemplo, em linhas de produção em que não existe espaço para constituir inventários intermédios entre máquinas, devendo uma máquina estar livre para processar a tarefa quando termina a operação na máquina precedente. Este tipo de problemas pode ser formulado como um probema de caixeiro viajante, apresentado na Secção 3.10.

Considere uma linha de laminagem em que cada tarefa é constituída pelas operações de fundição, vazamento e laminagem a quente. Neste caso, não pode haver esperas entre as operações de uma tarefa, pelo facto de ser necessário manter a temperatura ao longo de todo o processo. Considere o conjunto de 3 tarefas abaixo indicadas, a executar na linha de laminagem. Os tempos de processamento, p_{hi}, da $i^{ésima}$ operação da Tarefa h são os seguintes:

	Fundição	Vazamento	Laminagem
Tarefa 1	1	4	4
Tarefa 2	2	2	1
Tarefa 3	1	3	2

O tempo total necessário para completar todas as tarefas depende do modo como as tarefas são sequenciadas. No primeiro caso, em que a sequência é 123, o tempo necessário é de 14 unidades, enquanto que, para a sequência 132, o tempo necessário é de apenas 12 unidades, conforme apresentado nos seguintes diagramas de Gantt:





É fácil de verificar que o tempo total necessário é dado pela soma dos tempos de transição entre tarefas e da duração da última tarefa que é executada. A título ilustrativo, no primeiro caso, o tempo que medeia entre o início da tarefa 1 e o início da tarefa 2 é 5, o tempo que medeia entre o início da tarefa 2 e o início da tarefa 3 é 3 e a duração da tarefa 3 é 6, pelo que a duração do plano é de 14. No segundo caso, o tempo que medeia entre o início da tarefa 1 e o início da tarefa 3 é 5, o tempo que medeia entre o início da tarefa 3 e o início da tarefa 2 é 2 e a duração da tarefa 2 é 5, pelo que a duração do plano é de 12.

Claramente, os tempos que medeiam entre o início de duas tarefas dependem das durações das operações compõem cada uma das tarefas, que, relembre-se, devem ser executadas sem interrupções.

O problema de sequenciar as tarefas de modo a minimizar o tempo total necessário à execução de todas as tarefas pode ser formulado como um problema de caixeiro viajante. Considere custos de transição c_{ij} correspondentes ao tempo que é necessário mediar entre o início das tarefa i e o início da tarefa j, por esta ordem, de modo a assegurar as tarefas são executadas sem esperas intermédias.

A matriz dos custos de transição é a seguinte:

	1	2	3
1	–	5	5
2	3	–	3
3	3	2	–

Estas tarefas são realizadas uma única vez. Para a formulação do modelo, é necessário considerar uma tarefa extra, que representa o início e o fim dos trabalhos, para modelar a execução

cíclica das tarefas. Esta tarefa extra tem uma duração nula nas três operações, pelo que somos conduzidos à seguinte matriz de custos de uma problema de caixeiro viajante:

	extra	1	2	3
extra	–	0	0	0
1	9	–	5	5
2	5	3	–	3
3	6	3	2	–

A solução óptima deste problema tem um custo de 12 unidades, e corresponde ao seguinte ciclo: extra – 1 – 3 – 2 – extra. Atendendo ao modelo adoptado, o ciclo corresponde à realização das tarefas segundo a sequência 1 – 3 – 2.

5.14 Problema do carteiro chinês

Este problema consiste em determinar o percurso de comprimento mínimo que atravessa todos os arestas de um grafo, pelo menos uma vez. A primeira referência que lhe foi feita apareceu numa revista chinesa, em 1962, tendo a designação ficado associada ao problema.

Num problema de distribuição do correio, podemos associar a área atribuída a um carteiro a um grafo, em que as arestas representam as ruas e os vértices representam os cruzamentos. Para efectuar a distribuição do correio, o carteiro necessita de percorrer todas as ruas, pelo menos uma vez, e voltar, ao fim do dia, à estação de correios. Sabendo que a cada aresta está associado um custo (por exemplo, distância), o problema consiste em determinar o caminho que o carteiro deve percorrer para minimizar a distância total.

O problema do carteiro chinês pode ser formulado como o problema da determinação do circuito Euleriano de menor custo, num grafo em que existem custos associados às arestas. Para o resolver, é necessário recorrer à resolução de subproblemas de caminho mais curto entre cada par de vértices e de um subproblema de emparelhamento de custo mínimo em grafos não–bipartidos, com custos associados às arestas.

Notar a diferença entre este problema e o problema do caixeiro viajante. No segundo, é necessário visitar todos os vértices de um grafo, não sendo necessário percorrer todas as arestas.

Em primeiro lugar, vamos analisar circuitos Eulerianos de um grafo que estão na base do problema. Faz-se também a distinção entre circuitos e caminhos Eulerianos.

5.14.1 Caminhos e circuitos Eulerianos

O problema das pontes de Königsberg é reconhecido como sendo o primeiro problema de teoria de grafos. A solução deste problema foi apresentada por Euler em 1736.

O rio que atravessa esta cidade tem duas ilhas, designadas na Figura 5.6 por a e por d. Entre as margens e as duas ilhas, existem as 7 pontes indicadas. A questão colocada era se existia um percurso que atravessasse todos as pontes uma e uma só vez e que voltasse ao ponto de partida.

Se associarmos um vértice a cada região e unirmos os vértices com um número de arestas igual ao número de pontes entre as regiões, obtemos o multigrafo apresentado na Figura 5.6, que constitui um modelo do problema.

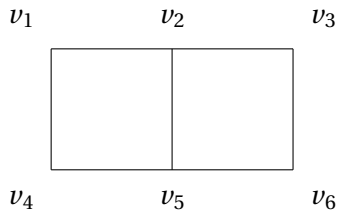
Definição 4. Um multigrafo $M = (V, A)$ não–orientado é uma estrutura em que pode existir mais de um aresta entre dois vértices.

Figura 5.6: Pontes de Königsberg

Iremos distinguir dois conceitos: circuitos eulerianos e caminhos eulerianos.

Definição 5. Um multigrafo ligado tem um caminho euleriano, se for possível seleccionar um vértice de partida e percorrer todas as arestas do multigrafo uma e uma só vez.

Exemplo 5.5. Para o multigrafo apresentado de seguida, o caminho definido pelos vértices $v_2, v_3, v_6, v_5, v_2, v_1, v_4$ e v_5 é um caminho euleriano.



Para este exemplo, o início do caminho deve ser o vértice v_2 (ou v_5). Se pretendermos atravessar todas as arestas do multigrafo, em particular os incidentes em v_2 (ou v_5), então uma destas arestas deve ser uma aresta terminal do caminho. □

Para um multigrafo possuir um caminho euleriano, é necessário que tenha, no máximo, dois vértices de grau ímpar. Neste caso particular, estes serão os vértices de partida e de chegada do caminho euleriano.

Teorema 4. *Num grafo (multigrafo), o número de vértices de grau ímpar é sempre par.*

Prova: Seja d_i o grau do vértice $i \in V$.

$$\sum_{i \in V} d_i = 2 \mid A \mid,$$

sendo $\mid A \mid$ o número de arestas do grafo. Vamos partir o conjunto de vértices em dois subconjuntos V_P, os de grau par, e V_I, os de grau ímpar.

$$\sum_{i \in V_P} d_i + \sum_{i \in V_I} d_i = 2 \mid A \mid .$$

Dado que o primeiro termo é par, bem como o resultado, o segundo termo $\sum_{i \in V_I} d_i$ deve ser necessariamente par. Como este termo é constituído por várias parcelas ímpares, o número de parcelas deve ser par. □

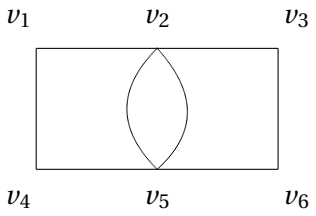
Definição 6. Um grafo é Euleriano se tiver um circuito euleriano.

Neste caso, qualquer que seja o vértice de partida seleccionado, é possível percorrer todas as arestas do multigrafo e voltar ao ponto de partida.

Teorema 5. Um grafo é Euleriano se e só se o grafo for ligado e o grau de todos os vértices for par.

Prova: A necessidade destas condições é trivial. O circuito Euleriano, para cada aresta de entrada no vértice, deve ter uma aresta de saída. É possível mostrar que a condição é também suficiente através da composição de circuitos (ver Berge). □

Exemplo 5.6. Para o multigrafo apresentado, o circuito definido pelos vértices $v_1, v_2, v_5, v_2, v_3, v_6, v_5, v_4$ e v_1 é um circuito Euleriano.



É possível mostrar que o multigrafo correspondente à cidade de Königsberg não possui nenhum caminho de Euler. De facto, o número de vértices de grau ímpar é 4. □

Note–se que, para um multigrafo qualquer, é muito fácil provar a não–existência de um caminho (ou de um circuito) de Euler; basta analisar o grau dos vértices e exibir um número de vértices de grau ímpar maior ou igual a 3 (ou maior ou igual a 1).

5.14.2 Formulação como problema de emparelhamento

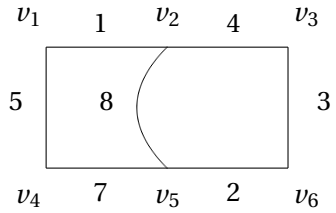
No problema do carteiro chinês, é dado um multigrafo com custos associados às arestas. O objectivo do problema é determinar o circuito de menor custo, atravessando todas as arestas, pelo menos uma vez. Para tornar possível a existência do circuito, pode ser necessário percorrer algumas das arestas mais de uma vez.

Teorema 6. Se o multigrafo for euleriano, qualquer circuito de Euler constitui uma solução óptima para o problema do carteiro chinês.

Prova: Um circuito de Euler é uma solução suficiente porque atravessa todas as arestas. É, também, a solução de menor custo, porque é necessário percorrer todas as arestas, pelo menos uma vez, incorrendo o respectivo custo. □

Por outro lado, se o multigrafo não for Euleriano, *i.e.*, se houver vértices de grau ímpar, é necessário percorrer algumas das arestas mais de uma vez. A selecção das arestas a percorrer mais de uma vez, far–se–á atendendo ao custo das arestas. Claramente, após a duplicação das arestas, o multigrafo resultante terá todos os vértices de grau par.

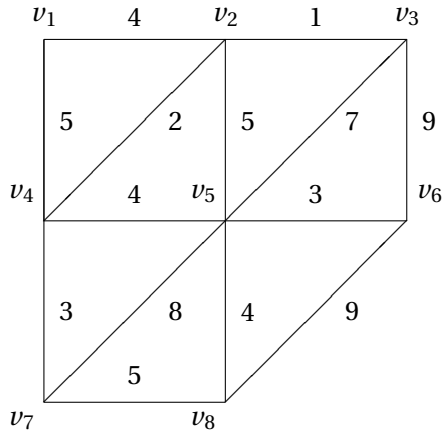
Exemplo 5.7. No seguinte Exemplo, existe apenas um par de vértices de grau ímpar. Para obter a solução óptima do problema, é necessário duplicar a aresta entre esses dois vértices v_2 e v_5, porque a aresta (v_2, v_5) corresponde ao caminho mais curto entre esses dois vértices.



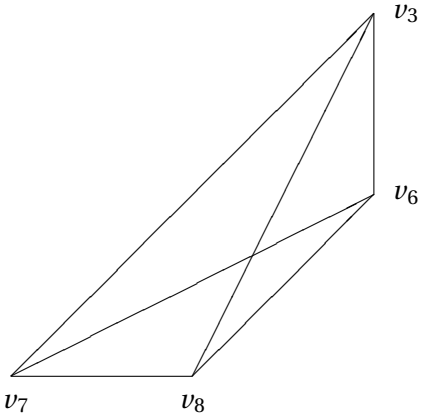
Após a duplicação, todos os vértices têm grau par, e o multigrafo resultante possui um circuito de Euler. □

Numa situação geral, o problema da duplicação de arestas pode ser resolvido como um problema de emparelhamento de menor custo num grafo não–bipartido. Este grafo será constituído pelos vértices do grafo original com grau ímpar.

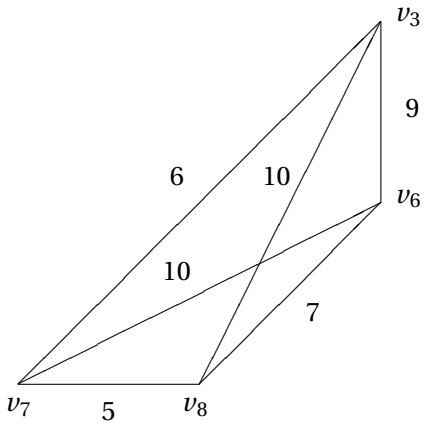
Exemplo 5.8. Considere o exemplo apresentado de seguida:



O conjunto de vértices de grau ímpar é $V_I = \{v_3, v_6, v_7, v_8\}$. Entre cada par de vértices de grau ímpar, existe um caminho.

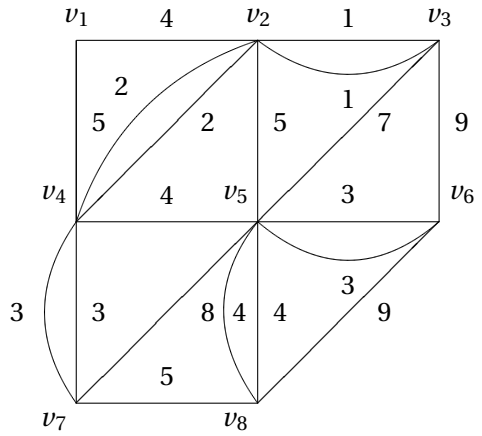


Existem várias duplicações de arestas que tornam todos os vértices de grau par. A enumeração completa produz as seguintes três soluções: $\{(v_3, v_6), (v_7, v_8)\}, \{(v_3, v_7), (v_6, v_8)\}$ e $\{(v_3, v_8), (v_6, v_7)\}$. Cada solução corresponde a um emparelhamento de cardinal 2, que acrescenta dois caminhos entre dois pares de vértices. Cada caminho deve corresponder ao caminho mais curto entre o par de vértices. A sua determinação produz os seguintes resultados:



O emparelhamento de menor custo é constituído pelas arestas (v_3, v_7) e (v_6, v_8), sendo o custo correspondente de 13. A aresta (v_3, v_7) do emparelhamento corresponde ao caminho mais curto entre esses dois vértices; esse caminho é constituído pelas arestas (v_3, v_2), (v_2, v_4) e (v_4, v_7). Por outro lado, a aresta (v_6, v_8) do emparelhamento corresponde às arestas (v_6, v_5) e (v_5, v_8).

A duplicação das arestas correspondentes aos caminhos mais curtos dá origem a um multigrafo com todos os vértices de grau par. Assim, a solução óptima para o problema do carteiro chinês será a indicada de seguida.



□

Outras aplicações

Entre outra possíveis aplicações, citam–se a recolha de lixo, a limpeza de neve das estradas e a inspecção de linha telefónicas. Em todos os casos, é necessário percorrer todas as arestas, pelo menos uma vez.

Notas

O problema do emparelhamento de menor custo para grafos não–bipartidos é o problema equivalente ao problema de afectação para grafos bipartidos. Existe um algoritmo polinomial, desenvolvido por Edmonds, para a resolução deste problema. Trata–se de um algoritmo primal–dual,

em que, em cada iteração, é resolvido um problema de emparelhamento máximo num grafo não–bipartido.

Existem outras versões deste problema para grafos orientados e para grafos mistos, que combinam arcos orientados e arestas não–orientadas. O problema em grafos orientados pode ser resolvido como um problema de transportes num grafo bipartido, em tempo polinomial. Por outro lado, o problema em grafos mistos é NP-completo.

5.15 Puzzles: Sudoku 2 × 2

O sudoku é um puzzle lógico em que se pretende preencher todas as células com algarismos, de forma a que não haja repetição de nenhum algarismo nas linhas, nas colunas ou nos blocos. Vamos ver uma versão com uma matriz 4×4, dividida em 4 blocos 2×2, em que os algarismos de algumas das células são dados. As regras do jogo impõem que:

- cada célula apenas pode ter um algarismo;
- numa linha, cada algarismo apenas pode aparecer uma vez;
- numa coluna, cada algarismo apenas pode aparecer uma vez;
- num bloco 2 × 2, cada algarismo apenas pode aparecer uma vez; e
- há células que devem ter o algarismo dado no puzzle.

Exemplo 5.9. O puzzle proposto tem os seguintes dados.

	1	4	
			2
1			
	3	2	

A problema pode ser formulado como um problema com variáveis de decisão binárias que indicam se o algarismo k é colocado na célula (i, j). A definição das variáveis de decisão é a seguinte:

$$x_{ijk} = \begin{cases} 1 & \text{, se a célula } (i,j) \text{ tiver o algarismo } k \\ 0 & \text{, caso contrário} \end{cases}$$

O objectivo do jogo é obter uma solução admissível, sendo todas as soluções igualmente boas. Portanto, a função objectivo pode ser uma função qualquer. O modelo é o seguinte:

```
/* funcão objectivo */
min: ;

/* restrições */
// cada célula apenas pode ter um algarismo
celula11: x111 + x112 + x113 + x114 = 1;
celula12: x121 + x122 + x123 + x124 = 1;
celula13: x131 + x132 + x133 + x134 = 1;
celula14: x141 + x142 + x143 + x144 = 1;
```

```
celula21: x211 + x212 + x213 + x214 = 1;
celula22: x221 + x222 + x223 + x224 = 1;
celula23: x231 + x232 + x233 + x234 = 1;
celula24: x241 + x242 + x243 + x244 = 1;

celula31: x311 + x312 + x313 + x314 = 1;
celula32: x321 + x322 + x323 + x324 = 1;
celula33: x331 + x332 + x333 + x334 = 1;
celula34: x341 + x342 + x343 + x344 = 1;

celula41: x411 + x412 + x413 + x414 = 1;
celula42: x421 + x422 + x423 + x424 = 1;
celula43: x431 + x432 + x433 + x434 = 1;
celula44: x441 + x442 + x443 + x444 = 1;

// numa linha, cada algarismo apenas pode aparecer uma vez
lin1num1: x111 + x121 + x131 + x141 = 1;
lin1num2: x112 + x122 + x132 + x142 = 1;
lin1num3: x113 + x123 + x133 + x143 = 1;
lin1num4: x114 + x124 + x134 + x144 = 1;

lin2num1: x211 + x221 + x231 + x241 = 1;
lin2num2: x212 + x222 + x232 + x242 = 1;
lin2num3: x213 + x223 + x233 + x243 = 1;
lin2num4: x214 + x224 + x234 + x244 = 1;

lin3num1: x311 + x321 + x331 + x341 = 1;
lin3num2: x312 + x322 + x332 + x342 = 1;
lin3num3: x313 + x323 + x333 + x343 = 1;
lin3num4: x314 + x324 + x334 + x344 = 1;

lin4num1: x411 + x421 + x431 + x441 = 1;
lin4num2: x412 + x422 + x432 + x442 = 1;
lin4num3: x413 + x423 + x433 + x443 = 1;
lin4num4: x414 + x424 + x434 + x444 = 1;

// numa coluna, cada algarismo apenas pode aparecer uma vez
col1num1: x111 + x211 + x311 + x411 = 1;
col1num2: x112 + x212 + x312 + x412 = 1;
col1num3: x113 + x213 + x313 + x413 = 1;
col1num4: x114 + x214 + x314 + x414 = 1;

col2num1: x121 + x221 + x321 + x421 = 1;
col2num2: x122 + x222 + x322 + x422 = 1;
col2num3: x123 + x223 + x323 + x423 = 1;
col2num4: x124 + x224 + x324 + x424 = 1;
```

```
col3num1: x131 + x231 + x331 + x431 = 1;
col3num2: x132 + x232 + x332 + x432 = 1;
col3num3: x133 + x233 + x333 + x433 = 1;
col3num4: x134 + x234 + x334 + x434 = 1;

col4num1: x141 + x241 + x341 + x441 = 1;
col4num2: x142 + x242 + x342 + x442 = 1;
col4num3: x143 + x243 + x343 + x443 = 1;
col4num4: x144 + x244 + x344 + x444 = 1;

// num bloco 2 x 2, cada algarismo apenas pode aparecer uma vez
NWnum1: x111 + x121 + x211 + x221 = 1;
NWnum2: x112 + x122 + x212 + x222 = 1;
NWnum3: x113 + x123 + x213 + x223 = 1;
NWnum4: x114 + x124 + x214 + x224 = 1;

NEnum1: x131 + x141 + x231 + x241 = 1;
NEnum2: x132 + x142 + x232 + x242 = 1;
NEnum3: x133 + x143 + x233 + x243 = 1;
NEnum4: x134 + x144 + x234 + x244 = 1;

SWnum1: x311 + x321 + x411 + x421 = 1;
SWnum2: x312 + x322 + x412 + x422 = 1;
SWnum3: x313 + x323 + x413 + x423 = 1;
SWnum4: x314 + x324 + x414 + x424 = 1;

SEnum1: x331 + x341 + x431 + x441 = 1;
SEnum2: x332 + x342 + x432 + x442 = 1;
SEnum3: x333 + x343 + x433 + x443 = 1;
SEnum4: x334 + x344 + x434 + x444 = 1;

// células que devem ter o algarismo dado no puzzle
dado12: x121 = 1;
dado13: x134 = 1;
dado24: x242 = 1;
dado31: x311 = 1;
dado42: x423 = 1;
dado43: x432 = 1;

// comentário:
// o lpsolve "confunde-se" quando se declara uma variável como
// binária depois de lhe atribuir explicitamente o valor 0 ou 1,
// como foi feito.
//
// declarar essas variáveis como binárias tem como efeito que
// a atribuição é ignorada.
```

```
bin
x111 x112 x113 x114        x122 x123 x124
x131 x132 x133        x141 x142 x143 x144
x211 x212 x213 x214 x221 x222 x223 x224
x231 x232 x233 x234 x241        x243 x244
      x312 x313 x314 x321 x322 x323 x324
x331 x332 x333 x334 x341 x342 x343 x344
x411 x412 x413 x414 x421 x422        x424
x431        x433 x434 x441 x442 x443 x444;
```

A solução do modelo, obtida com o LPSsolve é $x_{121} = x_{134} = x_{242} = x_{311} = x_{423} = x_{432} = 1$ (conforme a atribuição feita no modelo), $x_{112} = x_{143} = x_{213} = x_{224} = x_{231} = x_{322} = x_{333} = x_{344} = x_{414} = x_{441} = 1$, sendo todas as outras variáveis $x_{ijk} = 0$.

Assim a solução do puzzle é:

2	1	4	3
3	4	1	2
1	2	3	4
4	3	2	1

□

5.16 Lotes de produção multi-artigo

Uma empresa produz sumos de frutas, laranja, maçã e pêra, à base de concentrado. A produção consiste em misturar as matérias primas, concentrado de sumo de fruta, água e açúcar, e encher os pacotes de sumo na única linha de engarrafamento. O tempo de produção de um dado sumo corresponde ao tempo de engarrafamento, dado que a mistura das matérias primas é feita num tempo negligenciável. Há dois armazéns, o de matérias primas, onde são guardados os tambores com os concentrados de sumo, e o de produtos finais, que guarda as paletes com os pacotes de sumo.

Pretende-se determinar as quantidades a comprar, a produzir e a armazenar, em cada período e de cada tipo de sumo, de modo a fornecer os pedidos dos clientes num horizonte de planeamento de 12 meses, com um custo global mínimo.

As quantidades de concentrado serão expressas em unidades equivalentes (U.E.). Uma U.E. de concentrado de fruta é o peso de concentrado necessário para produzir uma tonelada de produto final (que equivale a uma U.E. de produto final).

O contrato com o cliente prevê a entrega das seguintes quantidades, em U.E.:

Mês	jan	fev	mar	abr	mai	jun	jul	ago	set	out	nov	dez
Período	1	2	3	4	5	6	7	8	9	10	11	12
Laranja	9	9	9	12	16	17	19	19	16	12	10	9
Maçã	5	5	5	6	8	9	10	10	8	6	5	5
Pêra	4	4	4	5	6	7	8	8	6	5	4	4

Os custos de produção [U.M./U.E.] são semelhantes para os três tipos de sumo, mas variam ao longo do tempo:

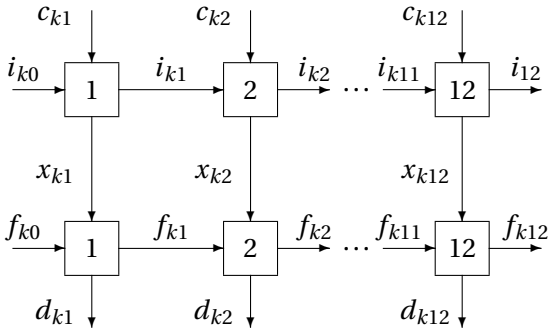


Figura 5.7: Rede do produto $k \in \{L, M, P\}$ do modelo de lotes de produção multiartigo

Mês	jan	fev	mar	abr	mai	jun	jul	ago	set	out	nov	dez
Período	1	2	3	4	5	6	7	8	9	10	11	12
custo de produção	10	10	10	10	10	10	10	12	10	10	10	10

A capacidade máxima de produção é de 40 U.E. por período. A capacidade máxima do armazém de tambores de concentrado de sumo é de 30 U.E.; há um custo de posse de inventário de 1 U.M./U.E.período. É possível fazer também inventário de paletes de pacotes de sumo, mas o custo de posse de inventário é de 3 U.M./U.E.período e a capacidade máxima de armazenagem é de 40 U.E.. No início, existem os seguintes inventários em armazém:

	laranja	maçã	pêra
armazém de matéria-prima (U.E.)	16	8	6
armazém de produto final (U.E.)	20	10	10

Pretende-se que, no fim do horizonte de planeamento, os inventários sejam os mesmos.

O preço do concentrado de fruta [U.M./U.E.] varia ao longo do ano, e é dado pela seguinte tabela:

Mês	jan	fev	mar	abr	mai	jun	jul	ago	set	out	nov	dez
Período	1	2	3	4	5	6	7	8	9	10	11	12
Laranja	145	145	155	150	165	160	170	145	150	135	125	145
Maçã	231	299	287	198	210	298	211	180	117	116	221	217
Pêra	116	116	124	120	132	128	136	116	120	108	100	116

Este problema que envolve 3 produtos pode ser representado por três redes do modelo de lotes de produção, cada uma associada a um produto diferente $k \in \{L, M, P\}$, designando L a laranja, M a maça e P a pera, como se mostra na Figura 5.7. A figura ilustra as restrições de conservação de fluxo, que dizem respeito a cada produto. Para além dessas restrições, há restrições relativas às capacidades de produção e dos inventários de matérias primas e de produtos acabados. Os 3 produtos diferentes partilham recursos, e é necessário determinar a que produtos devem ser atribuídos.

Esta formulação usa as Variáveis de decisão:

- c_{kj} : número de unidades compradas no período j, $j = 1, \ldots, 12$, do produto $k, k \in L, M, P$.

- x_{kj} : número de unidades produzidas no período $j, j = 1,\ldots,12$, do produto $k, k \in \{L,M,P\}$.
- i_{kj} : stock de matérias primas (inicial) existente após o período $j, j = 0,\ldots,12$, do produto $k, k \in \{L,M,P\}$.
- f_{kj} : stock de produtos acabados (final) existente após o período $j, j = 0,\ldots,12$, do produto $k, k \in \{L,M,P\}$.

```
////////////////////////////////////////////////////
//
//               Lotes de produção multi-artigo
//
////////////////////////////////////////////////////

Min: CustoLaranja + CustoMaca + CustoPera
      + 10 xL1 + 10 xL2 + 10 xL3 + 10 xL4 + 10 xL5 + 10 xL6
      + 10 xL7 + 12 xL8 + 10 xL9 + 10 xL10 + 10 xL11 + 10 xL12
      + 10 xM1 + 10 xM2 + 10 xM3 + 10 xM4 + 10 xM5 + 10 xM6
      + 10 xM7 + 12 xM8 + 10 xM9 + 10 xM10 + 10 xM11 + 10 xM12
      + 10 xP1 + 10 xP2 + 10 xP3 + 10 xP4 + 10 xP5 + 10 xP6
      + 10 xP7 + 12 xP8 + 10 xP9 + 10 xP10 + 10 xP11 + 10 xP12
      +  1 iL1 +  1 iL2 +  1 iL3 +  1 iL4 + 1 iL5 + 1 iL6
      +  1 iL7 + 1 iL8 + 1 iL9 + 1 iL10 + 1 iL11 + 1 iL12
      +  1 iM1 + 1 iM2 + 1 iM3 + 1 iM4 + 1 iM5 + 1 iM6
      +  1 iM7 + 1 iM8 + 1 iM9 + 1 iM10 + 1 iM11 + 1 iM12
      +  1 iP1 + 1 iP2 + 1 iP3 + 1 iP4 + 1 iP5 + 1 iP6
      +  1 iP7 + 1 iP8 + 1 iP9 + 1 iP10 + 1 iP11 + 1 iP12
      +  3 fL1 + 3 fL2 + 3 fL3 + 3 fL4 + 3 fL5 + 3 fL6
      +  3 fL7 + 3 fL8 + 3 fL9 + 3 fL10 + 3 fL11 + 3 fL12
      +  3 fM1 + 3 fM2 + 3 fM3 + 3 fM4 + 3 fM5 + 3 fM6
      +  3 fM7 + 3 fM8 + 3 fM9 + 3 fM10 + 3 fM11 + 3 fM12
      +  3 fP1 + 3 fP2 + 3 fP3 + 3 fP4 + 3 fP5 + 3 fP6
      +  3 fP7 + 3 fP8 + 3 fP9 + 3 fP10 + 3 fP11 + 3 fP12;

CustoLaranja = 145 cL1 + 145 cL2 + 155 cL3 + 150 cL4 + 165 cL5 + 160 cL6
               + 170 cL7 + 145 cL8 + 150 cL9 + 135 cL10+ 125 cL11+ 145 cL12;
CustoMaca    = 231 cM1 + 299 cM2 + 287 cM3 + 198 cM4 + 210 cM5 + 298 cM6
               + 211 cM7 + 180 cM8 + 117 cM9 + 116 cM10+ 221 cM11+ 217 cM12;
CustoPera    = 116 cP1 + 116 cP2 + 124 cP3 + 120 cP4 + 132 cP5 + 128 cP6
               + 136 cP7 + 116 cP8 + 120 cP9 + 108 cP10+ 100 cP11+ 116 cP12;

// Inventário de matéria prima
 iL0 = 16;
cL1 + iL0 - iL1 - xL1  = 0; cL2 + iL1 - iL2 - xL2  = 0;
cL3 + iL2 - iL3 - xL3  = 0; cL4 + iL3 - iL4 - xL4  = 0;
cL5 + iL4 - iL5 - xL5  = 0; cL6 + iL5 - iL6 - xL6  = 0;
cL7 + iL6 - iL7 - xL7  = 0; cL8 + iL7 - iL8 - xL8  = 0;
```

```
cL9 + iL8 - iL9 - xL9  = 0; cL10 + iL9 - iL10 - xL10 = 0;
cL11 + iL10 - iL11 - xL11 = 0; cL12 + iL11 - iL12 - xL12 = 0;
 iL12 = 16;

 iM0 = 8;
cM1 + iM0 - iM1 -xM1  = 0; cM2 + iM1 - iM2 -xM2  = 0;
cM3 + iM2 - iM3 -xM3  = 0; cM4 + iM3 - iM4 -xM4  = 0;
cM5 + iM4 - iM5 -xM5  = 0; cM6 + iM5 - iM6 -xM6  = 0;
cM7 + iM6 - iM7 -xM7  = 0; cM8 + iM7 - iM8 -xM8  = 0;
cM9 + iM8 - iM9 -xM9  = 0; cM10 + iM9 - iM10 -xM10  = 0;
cM11 + iM10 - iM11 - xM11 = 0; cM12 + iM11 - iM12 -xM12 = 0;
 iM12 = 8;

 iP0 = 6;
cP1 + iP0 - iP1 - xP1 = 0; cP2 + iP1 - iP2 - xP2  = 0;
cP3 + iP2 - iP3 - xP3 = 0; cP4 + iP3 - iP4- xP4  = 0;
cP5 + iP4 - iP5 - xP5 = 0; cP6 + iP5 - iP6 - xP6 = 0;
cP7 + iP6 - iP7 - xP7 = 0; cP8 + iP7 - iP8 - xP8 = 0;
cP9 + iP8 - iP9 - xP9 = 0; cP10 + iP9 - iP10 - xP10 = 0;
cP11 + iP10 - iP11 - xP11 = 0; cP12 + iP11 - iP12 - xP12 = 0;
iP12 = 6;

// Inventário final (produtos acabados)
 fL0 = 20;
xL1 + fL0 - fL1 = 9;  xL2 + fL1 - fL2 = 9;
xL3 + fL2 - fL3 = 9;  xL4 + fL3 - fL4 = 12;
xL5 + fL4 - fL5 = 16; xL6 + fL5 - fL6 = 17;
xL7 + fL6 - fL7 = 19; xL8 + fL7 - fL8 = 19;
xL9 + fL8 - fL9 = 16; xL10 + fL9 - fL10 = 12;
xL11 + fL10 - fL11 = 10; xL12 + fL11 - fL12 = 9;
fL12 = 20;

fM0 = 10;
xM1 + fM0 - fM1 = 5; xM2 + fM1 - fM2 = 5;
xM3 + fM2 - fM3 = 5; xM4 + fM3 - fM4 = 6;
xM5 + fM4 - fM5 = 8; xM6 + fM5 - fM6 = 9;
xM7 + fM6 - fM7 = 10; xM8 + fM7 - fM8 = 10;
xM9 + fM8 - fM9 = 8; xM10 + fM9 - fM10 = 6;
xM11 + fM10 - fM11 = 5; xM12 + fM11 - fM12 = 5;
fM12 = 10;

fP0 = 10;
xP1 + fP0 - fP1 = 4; xP2 + fP1 - fP2 = 4;
xP3 + fP2 - fP3 = 4; xP4 + fP3 - fP4 = 5;
xP5 + fP4 - fP5 = 6; xP6 + fP5 - fP6 = 7;
xP7 + fP6 - fP7 = 8; xP8 + fP7 - fP8 = 8;
xP9 + fP8 - fP9 = 6; xP10 + fP9 - fP10 = 5;
```

```
xP11 + fP10 - fP11 = 4; xP12 + fP11 - fP12 = 4;
fP12 = 10;

// Inventário máximo matéria prima:  30 unidades
iL1 + iM1 + iP1 <= 30; iL2 + iM2 + iP2 <= 30;
iL3 + iM3 + iP3 <= 30; iL4 + iM4 + iP4 <= 30;
iL5 + iM5 + iP5 <= 30; iL6 + iM6 + iP6 <= 30;
iL7 + iM7 + iP7 <= 30; iL8 + iM8 + iP8 <= 30;
iL9 + iM9 + iP9 <= 30; iL10 + iM10 + iP10 <= 30;
iL11 + iM11 + iP11 <= 30; iL12 + iM12 + iP12 <= 30;

// Inventário máximo final (produtos acabados):  40 unidades
fL1 + fM1 + fP1 <= 40; fL2 + fM2 + fP2 <= 40;
fL3 + fM3 + fP3 <= 40; fL4 + fM4 + fP4 <= 40;
fL5 + fM5 + fP5 <= 40; fL6 + fM6 + fP6 <= 40;
fL7 + fM7 + fP7 <= 40; fL8 + fM8 + fP8 <= 40;
fL9 + fM9 + fP9 <= 40; fL10 + fM10 + fP10 <= 40;
fL11 + fM11 + fP11 <= 40; fL12 + fM12 + fP12 <= 40;

// Capacidade máxima de produção: 30 unidades / período

xL1 + xM1 + xP1 <= 40; xL2 + xM2 + xP2 <= 40;
xL3 + xM3 + xP3 <= 40; xL4 + xM4 + xP4 <= 40;
xL5 + xM5 + xP5 <= 40; xL6 + xM6 + xP6 <= 40;
xL7 + xM7 + xP7 <= 40; xL8 + xM8 + xP8 <= 40;
xL9 + xM9 + xP9 <= 40; xL10 + xM10 + xP10 <= 40;
xL11 + xM11 + xP11 <= 40; xL12 + xM12 + xP12 <= 40;

////////////////////////////////////////////////
//      Apenas 1 produto por período
////////////////////////////////////////////////
/*
no1: yL1 + yM1 + yP1 <=1;
no2: yL2 + yM2 + yP2 <=1;
no3: yL3 + yM3 + yP3 <=1;
no4: yL4 + yM4 + yP4 <=1;
no5: yL5 + yM5 + yP5 <=1;
no6: yL6 + yM6 + yP6 <=1;
no7: yL7 + yM7 + yP7 <=1;
no8: yL8 + yM8 + yP8 <=1;
no9: yL9 + yM9 + yP9 <=1;
no10: yL10 + yM10 + yP10 <=1;
no11: yL11 + yM11 + yP11 <=1;
no12: yL12 + yM12 + yP12 <=1;

xL1 <= 40 yL1; xL2 <= 40 yL2; xL3 <= 40 yL3; xL4 <= 40 yL4;
xL5 <= 40 yL5; xL6 <= 40 yL6; xL7 <= 40 yL7; xL8 <= 40 yL8;
```

```
xL9 <= 40 yL9; xL10 <= 40 yL10; xL11 <= 40 yL11; xL12 <= 40 yL12;

xM1 <= 40 yM1; xM2 <= 40 yM2; xM3 <= 40 yM3; xM4 <= 40 yM4;
xM5 <= 40 yM5; xM6 <= 40 yM6; xM7 <= 40 yM7; xM8 <= 40 yM8;
xM9 <= 40 yM9; xM10 <= 40 yM10; xM11 <= 40 yM11; xM12 <= 40 yM12;

xP1 <= 40 yP1; xP2 <= 40 yP2; xP3 <= 40 yP3; xP4 <= 40 yP4;
xP5 <= 40 yP5; xP6 <= 40 yP6; xP7 <= 40 yP7; xP8 <= 40 yP8;
xP9 <= 40 yP9; xP10 <= 40 yP10; xP11 <= 40 yP11; xP12 <= 40 yP12;

bin yL1,yM1,yP1,yL2,yM2,yP2,yL3,yM3,yP3,yL4,yM4,yP4;
bin yL5,yM5,yP5,yL6,yM6,yP6,yL7,yM7,yP7,yL8,yM8,yP8;
bin yL9,yM9,yP9,yL10,yM10,yP10,yL11,yM11,yP11,yL12,yM12,yP12;
```

O custo da solução óptima é 46137. O modelo apresentado permite que, num dado período, sejam produzidos vários tipos de sumo. Se se pretender um plano de produção em que, em cada período, apenas se produza um único tipo de sumo, devem ser usadas as restrições no final do modelo, inibidas como comentário. Neste caso, o custo óptimo passa a ser 47651.

É também possível construir um modelo em que existe um custo associado a uma transição de produto. Por exemplo, se no período j houver produção de maçã e no período $j+1$ se transitar para a produção de laranja, ocorre um custo de mudança de produção, que pode resultar de operações de preparação, ou de outro tipo; se, no período $j+1$ se continuar a produção de maçã, então esse custo não ocorre. A construção deste modelo é deixado como desafio.

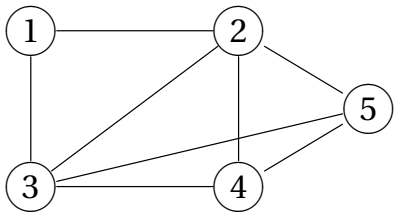
5.17 Redes de comunicação

As redes de comunicação podem ser representadas por um grafo, em que os arcos (linhas) representam linhas de transmissão e os vértices (nós) representam pontos de origem e de destino de mensagens, e/ou *routers*, que as encaminham para outros nós da rede. O grafo da rede de comunicação $G = (V, A)$ é constituído por um conjunto de vértices V e um conjunto de arcos A.

As redes são concebidas para permitir o estabelecimento de comunicações entre cada par de nós, simultaneamente. Existe um conjunto de fluxos de tráfego distintos, γ_{st}, cada um deles identificado por uma origem s e um destino $t, \forall s, t \in V$. Cada um destes fluxos é designado por *stream*. Neste caso, vamos considerar um número restrito de *streams*, o que não tira generalidade à análise, porque os modelos com todas as *streams* seriam idênticos, e apenas maiores.

Há estratégias em que o encaminhamento do tráfego de uma *stream* é todo feito pelo mesmo caminho. Iremos analisar esse caso no Trabalho 3. Neste trabalho, vamos considerar uma estratégia de encaminhamento multi-caminho (também designado por proporcional). O *router*, existente num nó da rede, identifica o destino de cada mensagem que recebe e encaminha-a por uma de várias linhas alternativas, previamente definidas, de acordo com uma tabela que estabelece qual a percentagem de tráfego a encaminhar por cada uma dessas linhas alternativas consoante o destino. Se se combinarem as diferentes alternativas em cada nó, esta estratégia dá origem a um conjunto de caminhos distintos para cada *stream*, partilhando evidentemente a mesma origem s e o mesmo destino t.

Considere a rede de comunicações com 5 vértices apresentada na figura:



A matriz de tráfego $\Gamma = [\gamma_{st}]$ é a seguinte:

$$\Gamma = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ 1 & & 8 & & 30 & 20 \\ 2 & & & 12 & 16 & \\ 3 & & & & 4 & 3 \\ 4 & & & & & 10 \\ 5 & & & & & \end{array}$$

Os problemas em redes podem ser modelados usando diferentes conjuntos de variáveis de decisão, por exemplo, variáveis de decisão que representam caminhos ou que estão associadas ao fluxo que atravessa os cortes do grafo. Neste trabalho, deve ser usado um modelo de fluxos em arcos, em que cada variável de decisão representa a quantidade de fluxo num arco de uma *stream*, ou seja, x_{ij}^k, que designa o fluxo no arco $(i, j), \forall (i, j) \in A$, da *stream* $k, \forall k \in K$, sendo K o conjunto de *streams*.

Os custos unitários de utilização da linha de transmissão (i, j), designados por c_{ij}, sendo $c_{ij} = c_{ji}, \forall (i, j) \in A$, são os apresentados na seguinte matriz $C = [c_{ij}]$:

$$C = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ 1 & & 8 & 12 & & \\ 2 & & & 2 & 7 & 10 \\ 3 & & & & 4 & 6 \\ 4 & & & & & 1 \\ 5 & & & & & \end{array}$$

Existem capacidades máximas associadas a cada linha de transmissão. Os fluxos de tráfego que usam uma dada linha de transmissão partilham os recursos disponíveis: a soma dos fluxos de tráfego que circulam em ambos os sentidos da linha não pode exceder a capacidade máxima da linha. A capacidade máxima é igual para todas as linhas $u_{ij} = 32, \forall i, j$.

Pretende-se determinar como deveria ser efectuado o encaminhamento de mensagens de modo a minimizar os custos de operação.

```
/*
  Problem name: multicommodity flow problem: arc flow model
*/
min:   + 8x121 + 12x131 + 2x231 + 7x241 + 10x251 + 4x341 + 6x351 + 1x451
       + 8x211 + 12x311 + 2x321 + 7x421 + 10x521 + 4x431 + 6x531 + 1x541
       + 8x122 + 12x132 + 2x232 + 7x242 + 10x252 + 4x342 + 6x352 + 1x452
```

```
      + 8x212 + 12x312 + 2x322 + 7x422 + 10x522 + 4x432 + 6x532 + 1x542
      + 8x123 + 12x133 + 2x233 + 7x243 + 10x253 + 4x343 + 6x353 + 1x453
      + 8x213 + 12x313 + 2x323 + 7x423 + 10x523 + 4x433 + 6x533 + 1x543
      + 8x124 + 12x134 + 2x234 + 7x244 + 10x254 + 4x344 + 6x354 + 1x454
      + 8x214 + 12x314 + 2x324 + 7x424 + 10x524 + 4x434 + 6x534 + 1x544
      + 8x125 + 12x135 + 2x235 + 7x245 + 10x255 + 4x345 + 6x355 + 1x455
      + 8x215 + 12x315 + 2x325 + 7x425 + 10x525 + 4x435 + 6x535 + 1x545
      + 8x126 + 12x136 + 2x236 + 7x246 + 10x256 + 4x346 + 6x356 + 1x456
      + 8x216 + 12x316 + 2x326 + 7x426 + 10x526 + 4x436 + 6x536 + 1x546
      + 8x127 + 12x137 + 2x237 + 7x247 + 10x257 + 4x347 + 6x357 + 1x457
      + 8x217 + 12x317 + 2x327 + 7x427 + 10x527 + 4x437 + 6x537 + 1x547
      + 8x128 + 12x138 + 2x238 + 7x248 + 10x258 + 4x348 + 6x358 + 1x458
      + 8x218 + 12x318 + 2x328 + 7x428 + 10x528 + 4x438 + 6x538 + 1x548;


// volume of stream
stream1 =  8;
stream2 = 30;
stream3 = 20;
stream4 = 12;
stream5 = 16;
stream6 =  4;
stream7 =  3;
stream8 = 10;

// flow conservation for stream 1 with (origin,destination) =  (1,2)
Stream1Vertice1:-x211-x311+x121+x131 = +stream1;
Stream1Vertice2:-x121-x321-x421-x521+x211+x231+x241+x251=-stream1;
Stream1Vertice3:-x131-x231-x431-x531+x311+x321+x341+x351=0 ;
Stream1Vertice4:-x241-x341-x541+x431+x421+x451=0;
Stream1Vertice5:-x251-x351-x451+x521+x531+x541=0;

// flow conservation for stream 2 with (origin,destination) =  (1,4)
Stream2Vertice1:-x212-x312+x122+x132 = +stream2;
Stream2Vertice2:-x122-x322-x422-x522+x212+x232+x242+x252=0;
Stream2Vertice3:-x132-x232-x432-x532+x312+x322+x342+x352=0;
Stream2Vertice4:-x242-x342-x542+x432+x422+x452=-stream2;
Stream2Vertice5:-x252-x352-x452+x522+x532+x542=0;

// flow conservation for stream 3 with (origin,destination) =  (1,5)
Stream3Vertice1:-x213-x313+x123+x133 =+stream3;
Stream3Vertice2:-x123-x323-x423-x523+x213+x233+x243+x253=0;
Stream3Vertice3:-x133-x233-x433-x533+x313+x323+x343+x353=0;
Stream3Vertice4:-x243-x343-x543+x433+x423+x453=0;
Stream3Vertice5:-x253-x353-x453+x523+x533+x543=-stream3;

// flow conservation for stream 4 with (origin,destination) =  (2,3)
```

```
Stream4Vertice1:-x214-x314+x124+x134 = 0;
Stream4Vertice2:-x124-x324-x424-x524+x214+x234+x244+x254=+stream4;
Stream4Vertice3:-x134-x234-x434-x534+x314+x324+x344+x354=-stream4;
Stream4Vertice4:-x244-x344-x544+x434+x424+x454=0;
Stream4Vertice5:-x254-x354-x454+x524+x534+x544=0;

// flow conservation for stream 5 with (origin,destination) =  (2,4)
Stream5Vertice1:-x215-x315+x125+x135 = 0;
Stream5Vertice2:-x125-x325-x425-x525+x215+x235+x245+x255=+stream5;
Stream5Vertice3:-x135-x235-x435-x535+x315+x325+x345+x355=0;
Stream5Vertice4:-x245-x345-x545+x435+x425+x455=-stream5;
Stream5Vertice5:-x255-x355-x455+x525+x535+x545=0;

// flow conservation for stream 6 with (origin,destination) =  (3,4)
Stream6Vertice1:-x216-x316+x126+x136 = 0;
Stream6Vertice2:-x126-x326-x426-x526+x216+x236+x246+x256=0;
Stream6Vertice3:-x136-x236-x436-x536+x316+x326+x346+x356=+stream6 ;
Stream6Vertice4:-x246-x346-x546+x436+x426+x456=-stream6;
Stream6Vertice5:-x256-x356-x456+x526+x536+x546=0;

// flow conservation for stream 7 with (origin,destination) =  (3,5)
Stream7Vertice1:-x217-x317+x127+x137 = 0;
Stream7Vertice2:-x127-x327-x427-x527+x217+x237+x247+x257=0;
Stream7Vertice3:-x137-x237-x437-x537+x317+x327+x347+x357=+stream7;
Stream7Vertice4:-x247-x347-x547+x437+x427+x457=0;
Stream7Vertice5:-x257-x357-x457+x527+x537+x547=-stream7;

// flow conservation for stream 8 with (origin,destination) =  (4,5)
Stream8Vertice1:-x218-x318+x128+x138 = 0;
Stream8Vertice2:-x128-x328-x428-x528+x218+x238+x248+x258=0;
Stream8Vertice3:-x138-x238-x438-x538+x318+x328+x348+x358=0;
Stream8Vertice4:-x248-x348-x548+x438+x428+x458=+stream8;
Stream8Vertice5:-x258-x358-x458+x528+x538+x548=-stream8;

// capacity of arcs
Arco12: + x121 + x122 + x123 + x124 + x125 + x126 + x127 + x128
+ x211 + x212 + x213 + x214 + x215 + x216 + x217 + x218 <= 32;
Arco13: + x131 + x132 + x133 + x134 + x135 + x136 + x137 + x138
 + x311 + x312 + x313 + x314 + x315 + x316 + x317 + x318 <= 32;
Arco23: + x231 + x232 + x233 + x234 + x235 + x236 + x237 + x238
 + x321 + x322 + x323 + x324 + x325 + x326 + x327 + x328 <= 32;
Arco24: + x241 + x242 + x243 + x244 + x245 + x246 + x247 + x248
 + x421 + x422 + x423 + x424 + x425 + x426 + x427 + x428 <= 32;
Arco25: + x251 + x252 + x253 + x254 + x255 + x256 + x257 + x258
 + x521 + x522 + x523 + x524 + x525 + x526 + x527 + x528 <= 32;
Arco34: + x341 + x342 + x343 + x344 + x345 + x346 + x347 + x348
 + x431 + x432 + x433 + x434 + x435 + x436 + x437 + x438 <= 32;
```

```
Arco35: + x351 + x352 + x353 + x354 + x355 + x356 + x357 + x358
 + x531 + x532 + x533 + x534 + x535 + x536 + x537 + x538 <= 32;
Arco45: + x451 + x452 + x453 + x454 + x455 + x456 + x457 + x458
 + x541 + x542 + x543 + x544 + x545 + x546 + x547 + x548 <= 32;
```

A solução óptima do problema é dada no seguinte quadro (tendo todas as variáveis não mostradas o valor 0):

Variables	result
	1038
stream2	30
x122	24
x242	20
x133	20
x353	20
stream3	20
x235	16
stream5	16
x345	16
x234	12
stream4	12
x458	10
stream8	10
x342	9,99999999999999
stream1	8
x121	8
x132	6
stream6	4
x346	4
x232	3,99999999999999
stream7	3
x347	2,00000000000001
x457	2,00000000000001
x357	0,99999999999999
x131	0
x231	0
x241	0
x251	0

Anexo A

Complexidade Computacional (não é matéria da UC)

No início dos anos sessenta do século XX, aparecia na literatura o método dos planos de corte de Gomory, baseado em programação linear. Gomory provou que o método convergia num número finito de iterações, fornecendo a solução óptima do problema de programação inteira. Edmonds investigava na altura um problema relacionado com o problema de emparelhamento em grafos não–bipartidos, e sentiu que não era suficiente que um algoritmo fosse finito, que era necessário que ele fosse também um "bom"algoritmo, no sentido de eficiente ou polinomial, pois essa seria a única forma de conseguir resolver problemas de grande dimensão.

Edmonds conjecturou também que podia haver uma classe de problemas, como, por exemplo, o problema do caixeiro viajante e o problema geral de programação inteira, para os quais não pudessem ser desenvolvidos "bons"algoritmos. De facto, uma grande variedade de problemas pertencentes a esta classe tem sido objecto de análise por investigadores ao longo das últimas décadas, não tendo sido descoberto nenhum algoritmo eficiente de solução para nenhum deles. Existem boas razões para supôr que não podem ser resolvidos em tempo polinomial. Esta conjectura de Edmonds, que ainda permanece em aberto, será adiante formalizada em termos de classes de complexidade.

A teoria da complexidade é usada para analisar o grau de dificuldade de muitos problemas provenientes de áreas tão diversas como teoria de grafos, automata e linguagens, sequenciação e planeamento, jogos e puzzles e concepção de redes, e também para estabelecer classes de equivalência agregando problemas que têm o mesmo grau de complexidade.

A.1 Introdução

A dificuldade inerente à solução de um problema pode variar muito. O problema de afectação e o problema do caminho mais curto são exemplos de problemas que podem ser classificados, informalmente, como fáceis. Existem algoritmos eficientes que permitem determinar a solução destes problemas num tempo que é uma função polinomial da dimensão do problema, e, dado que estas funções crescem lentamente, é possível resolver problemas de dimensão relativamente grande.

Por outro lado, o problema do caixeiro viajante é um exemplo de um problema difícil. Para este problema, que tem vindo a ser investigado ao longo de já várias décadas, ainda não foi descoberto nenhum algoritmo eficiente para a sua solução, sendo de crer que tal algoritmo não exista. Para

determinar a solução óptima, é necessário recorrer a esquemas de enumeração, não sendo possível evitar, nalgumas situações, a enumeração de um número exponencial de soluções. Embora alguns algoritmos possam fornecer rapidamente a solução na maior parte dos casos, existem situações em que esses algoritmos necessitariam de tempos de execução que podem ser, mesmo utilizando processadores muito rápidos, da ordem dos séculos.

A teoria da complexidade classifica os problemas de acordo com a sua dificuldade intrínseca. Esta medida permite avaliar qual o tempo que é necessário para obter a solução de um problema. Noutra perspectiva, para um determinado problema, permite avaliar qual é a dimensão máxima que é possível solucionar num dado período de tempo.

Os problemas em que iremos concentrar a nossa atenção são os que pertencem à classe NP. O que caracteriza esta classe é a existência de um certificado de solução conciso que pode ser verificado em tempo polinomial.

A.2 Problemas e Instâncias de um Problema

Um problema pode ser apresentado em diferentes versões, como um problema de optimização ou como um problema de decisão. Conforme será analisado, é possível transformar um problema de optimização numa sequência de problemas de decisão, e estabelecer relações entre a dificuldade de solução de cada um deles. Normalmente, por questões de facilidade de tratamento, a complexidade de um problema é definida para a versão de decisão.

Um problema de decisão, P, é definido por um tipo de objecto (*e.g.*, grafo, conjunto, função), geralmente possuindo vários parâmetros, e por uma questão sobre uma propriedade desse objecto, cuja resposta pode ser "sim"ou "não". A um problema podem ser associadas instâncias, geralmente em número infinito, cada uma delas resultando de uma escolha específica de valores para os parâmetros do problema.

Exemplo A.1. Um problema de decisão é, por exemplo, o problema de decidir se um grafo é ou não bipartido. Este problema de decisão pode ser formalizado do seguinte modo: dado um grafo $G = (V, A)$, será que G é bipartido? A instância pode ser representada pela respectiva lista de arcos ou por uma qualquer representação alternativa de grafos não–orientados. □

A complexidade é expressa em função da dimensão dos dados. Assim, a forma de representar os dados assume particular relevância. Para qualquer instância, os dados devem ser representados usando sempre o mesmo esquema de codificação, que deve ser razoável, conciso e conter apenas a informação indispensável para a definição da instância. Define-se dimensão de uma instância como o comprimento necessário para uma codificação razoável dos dados.

Os dados de um problema devem ser representados utilizando símbolos de um alfabeto finito, designado por Σ. Uma sequência de símbolos (ou letras) é designada por *string*. Na base binária, utilizada em computadores, o alfabeto é $\Sigma = \{0, 1\}$. Com estes símbolos, eventualmente usando símbolos separadores, pode-se representar qualquer informação.

Exemplo A.2. Para o problema de decisão da bipartição de um grafo, uma representação adequada seria a listagem dos arcos do grafo, indicando os vértices em que cada arco é incidente. Para instâncias de pequena dimensão, como a apresentada, a representação de cada vértice pode ser feita utilizando, por exemplo, uma letra do alfabeto.

O número máximo de arcos de um grafo pode ser expresso como uma função polinomial do número de vértices. É fácil mostrar que, usando a codificação apresentada, a dimensão da instân-

	10	20	50	100	200	500	1000
n	0.00001	0.00002	0.00005	0.0001	0.0002	0.0005	0.001
$n\log n$	0.00003	0.00009	0.00028	0.00066	0.0015	0.0045	0.01
n^2	0.0001	0.0004	0.0025	0.01	0.04	0.25	1
n^3	0.001	0.008	0.125	1	8	125	17 min.
2^n	0.001	1.05	35 anos	4×10^{14}séc.	—	—	—
$n!$	3.63	771 séc.	1×10^{49}séc.	—	—	—	—

Tabela A.1: Tempos (em segundos, excepto onde indicado) requeridos por algoritmos de diferente complexidade

cia também poderá ser expressa por uma função polinomial com respeito ao número de vértices do grafo. Trata-se de uma representação concisa. □

A.3 Algoritmos

Um algoritmo é um procedimento constituído por uma sequência finita de instruções, destinado a obter a solução de um problema. A sequência de instruções deve permanecer inalterada ao longo do processo de obtenção da solução. O algoritmo deve necessariamente parar após um número finito de passos, produzindo a solução para qualquer instância do problema.

Sendo ponto assente que um algoritmo deve ser um procedimento finito, é importante diferenciar os procedimentos que terminam após um número de passos que pode ser expresso por uma função polinomial com respeito à dimensão da instância e aqueles para os quais não é possível estabelecer nenhum limite polinomial, pelo que são designados por exponenciais. Algoritmos polinomiais são aceitáveis, enquanto os algoritmos exponenciais não são aceitáveis.

Os algoritmos polinomiais de grau elevado não são úteis do ponto de vista prático. Por exemplo, um algoritmo que requeresse n^{100} operações não seria com certeza eficiente. No entanto, têm-se verificado que a barreira mais difícil de ultrapassar é a obtenção de um algoritmo polinomial; uma vez descoberto, o uso de estruturas de dados mais eficientes permite, geralmente, diminuir o grau do polinómio.

A diferença entre o desempenho dos algoritmos polinomiais e dos exponenciais pode ser ilustrada através dos tempos requeridos para a sua execução num computador. Se considerarmos que cada operação é efectuada num microsegundo, o tempo correspondente a diversas funções de n é o indicado no Quadro A.1.

O desenvolvimento tecnológico traduz-se num aumento das velocidades de computação, permitindo resolver, durante um dado tempo de computação, instâncias de maior dimensão. No entanto, o crescimento da dimensão das instâncias que é possível resolver não é igual para todos os casos. O Quadro A.2 apresenta a dimensão das instâncias que é possível resolver numa hora, sendo a velocidade de processamento actual de 1 operação por microsegundo, e em sistemas 10 e 100 vezes mais rápidos, respectivamente.

Enquanto, para algoritmos polinomiais, é possível resolver instâncias de dimensão de uma ordem de grandeza maior, no caso de algoritmos exponenciais, apenas é possível resolver instâncias que são algumas unidades maiores. O crescimento da ordem de grandeza da velocidade de computação, nos moldes descritos, não permitirá, só por si, dar resposta a este problema.

Na teoria da complexidade, os problemas são classificados em classes entre as quais é possível estabelecer uma hierarquia de graus de dificuldade. Há desde uma classe de problemas "simples",

	1	×10	×100
n	3.6×10^{9}	3.6×10^{10}	3.6×10^{11}
$n\log_2 n$	1.3×10^{8}	1.2×10^{9}	1.1×10^{10}
n^2	6.0×10^{4}	1.9×10^{5}	6.0×10^{5}
n^3	1.5×10^{3}	3.3×10^{3}	7.1×10^{3}
2^n	31	35	38
$n!$	12	13	14

Tabela A.2: Dimensão das instâncias que é possível resolver numa hora com diferentes velocidades de computação

para os quais é possível exibir um algoritmo que fornece uma solução em tempo polinomial, até problemas para os quais não podem sequer ser construídos algoritmos de solução. Estes problemas são classificados como indecidíveis. Pertencem a esta classe, por exemplo, o problema da paragem de Turing. Para os problemas desta classe, não é possível conceber um algoritmo que, numa sequência finita de passos, dê uma resposta, para todas as instâncias, à questão colocada.

Exemplo A.3. Em termos de computadores, o problema da paragem de Turing consiste em determinar se um dado programa de computador pára, ou não. Para responder a esta questão, seria necessário conceber um programa que recebesse como *input* o programa de computador em análise e que, após um número finito de passos decidisse se o programa pára, ou não.

Pode haver um programa que analise linha a linha o programa dado e possa estabelecer, nalguns casos, que o programa termina num número finito de passos e, noutros casos, que o programa não pára, porque, por exemplo, possui um ciclo onde permanece infinitamente. No entanto, não é possível obter um programa geral que responda cabalmente a esta questão para todos os possíveis programas de computador. □

A.3.1 Representação dos Dados do Problema

Em nenhum caso deve ser possível desviar a dificuldade inerente ao problema para a dimensão da instância. Para servir de ilustração, considere as seguintes duas representações alternativas do problema de optimização do caixeiro viajante: uma forma concisa, contendo a indicação do grafo e dos custos que lhe estão associados, ou, uma representação alternativa, consistindo na formulação de programação linear baseada no problema de afectação e contendo todas as restrições de eliminação de subcircuitos. Considera-se que a segunda forma de representar este problema não é aceitável. De facto, a utilização de programação linear permitiria determinar o óptimo de uma forma eficiente, em função da dimensão dos dados do problema. A questão é que esse algoritmo seria eficiente relativamente a uma representação que é exponencial.

Exemplo A.4. A representação de numeros inteiros em computadores é feita utilizando a base binária. O comprimento da *string* necessário à representação do número x é dado por $\lceil \log_2(x+1) \rceil$. Por exemplo, o número que é representado na base 10 por 31 tem uma representação binária que é 11111. □

Existem outras bases diferentes, como a decimal ou hexadecimal, que constituem formas alternativas de representação. As dimensões destas representações, quando se consideram bases maiores que a binária, são da mesma ordem de grandeza. O comprimento da *string* necessário à

representação do número na base B é dado por:

$$\log_B n = \frac{\log n}{\log B}.$$

Sendo $\log B$ uma constante, ambas dimensões são da mesma ordem de grandeza.

Existe ainda a base unária, que tem apenas interesse teórico, mas é usada na análise de complexidade de problemas pseudo-polinomiais. Nesta base, existe apenas um símbolo para fazer a representação dos números, pelo que esse símbolo deve ser repetido um número de vezes igual ao valor do número a representar. Na realidade, é esta a base usada para representar números quando, na infância, fazemos somas com os dedos.

Exemplo A.5. Considere que o símbolo disponível para a representação na base unária era o símbolo "/". O comprimento da *string* necessário à representação do número x é dado por x. Por exemplo, o número que é representado na base 10 por 12 tem uma representação unária que é ////////////. □

Do ponto de vista de dimensão, todas as representações em bases superiores à base binária são equivalentes, conforme visto. A representação unária necessita de um espaço que é exponencialmente maior do que o necessário para as outras representações.

A teoria da complexidade aplica-se a problemas que podem ser representados por dados inteiros ou racionais. Para um procedimento ser eficiente, os números gerados ao longo do processo devem manter-se dentro de limites razoáveis para poder garantir que as operações efectuadas sobre eles são executadas em tempo unitário. Se os números se tornarem exponencialmente grandes, será necessário um tempo exponencial para o seu processamento. Pela mesma razão, a teoria aqui exposta não se aplica a números reais para os quais não existe uma representação concisa. A representação destes números tem de ser feita recorrendo a aproximações.

A.3.2 Eficiência de Algoritmos

Há todo o interesse em tornar a avaliação de um algoritmo independente da máquina em que é executado. De facto, a teoria da complexidade classifica os algoritmos segundo uma função que exprime a ordem de grandeza do tempo que é necessário para a solução do problema. Esta medida da eficiência é expressa em função da dimensão da instância.

Para simplificar o cálculo da eficiência, considera-se que cada passo do algoritmo, quer seja uma operação aritmética, quer seja uma operação lógica ou qualquer outra, é executado numa unidade de tempo. A análise do algoritmo permite avaliar qual o número de passos que é necessário executar antes de terminar e produzir a solução desejada. Normalmente, o número de passos depende da dimensão da instância do problema e pode ser expresso como uma função dessa dimensão (*e.g.*, $12 \times 2^n + 45, 20 \times n^3 + 5 \times n^2, 4 \times n \log_2 n$).

Para a definição da eficiência, interessa sobretudo a ordem de grandeza da razão de crescimento do maior termo, porque, para valores suficientemente grandes, os termos com menores razões de crescimento se tornam desprezáveis; o mesmo acontece a constantes que multipliquem a maior razão de crescimento, que não afectam a ordem de grandeza. Formalmente, define-se $f(n) = O(g(n))$ se existir uma constante c tal que, para n suficientemente grande, se verifique

$$f(n) \leq c g(n).$$

Diz-se então, por exemplo, que um algoritmo é $O(n^2)$ (leia-se, de ordem n^2) ou exponencial, de ordem $O(2^n)$.

```
input: Lista L com n elementos
output: Lista L ordenada por ordem crescente
begin
    for i=1, n-1 do
        for j=i+1, n do
            if L(j) < L(i) then
                begin
                    temp=L(j);
                    L(j)=L(i);
                    L(i)=temp;
                end;
end;
```

Figura A.1: Algoritmo de Ordenação

Exemplo A.6. Considere o algoritmo de ordenação de uma lista, designado na literatura anglo-saxónica por *bubble sort*, apresentado na Figura A.1.

Dada uma lista de n elementos, o algoritmo executa trocas entre pares de elementos e produz a lista ordenada por ordem crescente de valor. O algoritmo tem dois ciclos encadeados, cada um dos quais é executado um número de vezes que é da ordem de grandeza de n. O número total de iterações é $O(n^2)$.

Em cada iteração, é efectuada uma comparação e, no pior dos casos, três atribuições de valor. O número de operações efectuadas em cada iteração não depende da dimensão da lista, mas é limitado por uma constante. Assim, a eficiência do algoritmo é $O(n^2)$. □

Existem algoritmos mais eficientes de ordenação, que não iremos aqui analisar, (*e.g.*, *mergesort, heapsort*) de eficiência $O(n \log_2 n)$.

Complexidade no Pior Caso

A sequência de passos executada pelo algoritmo depende dos dados da instância. Quando se refere complexidade, pretende-se normalmente designar a complexidade no pior caso. A complexidade de um problema é uma medida do tempo necessário para obter a solução da instância do problema que corresponde ao pior caso possível.

Evidentemente, esta medida de pior caso pode representar uma estimativa muito conservadora, podendo não corresponder à dificuldade prática em resolver a generalidade das instâncias, e ocorrendo apenas para alguns grupos de instâncias do problema. No entanto, só a medida daquela que é a situação mais desfavorável permite antecipadamente dar a garantia que o algoritmo irá produzir a solução da instância que pretendemos solucionar.

Complexidade Média

Outra medida do desempenho é a complexidade média. Esta medida permite classificar o desempenho esperado do algoritmo na resolução do problema, e pode ser bastante inferior à complexidade do pior caso. É de salientar que não é fácil estabelecer analiticamente resultados deste tipo para a generalidade dos casos. Há, no entanto, exemplos para alguns algoritmos de ordenação. O algoritmo de *quicksort* tem um desempenho médio $\mathrm{O}(n \log_2 n)$, embora no pior caso o problema possa requerer $\mathrm{O}(n^2)$ passos.

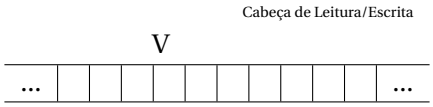


Tabela A.3: Máquina de Turing

O desempenho médio de um algoritmo com pior complexidade no pior caso pode ser melhor para a maior parte das situações práticas. A título ilustrativo, pode-se fazer uma comparação entre os desempenhos do algoritmo de simplex e dos algoritmos de ponto interior, que surgiram nos anos 80, destinados à solução do problema de programação linear.

O problema de programação linear é um problema polinomial, porque existem algoritmos de ponto interior que o resolvem em tempo polinomial. O algoritmo simplex com selecção da variável com custo reduzido mais negativo é exponencial, porque é possível construir instâncias, para qualquer dimensão, em que o algoritmo precisa de enumerar um conjunto exponencial de vértices antes de atingir a solução óptima.

Do ponto de vista prático, no momento actual, os dois algoritmos são competitivos, dependendo o seu desempenho das características das instância a resolver e não tendo nenhum dos dois adquirido a supremacia para a generalidade dos casos.

A.3.3 Máquina de Turing Determinística

O modelo de máquina que iremos apresentar foi proposto por Turing para resolver problemas de decisão e pode ser considerado como um percursor dos computadores actuais, que se baseiam no modelo da máquina RAM. Embora seja um modelo muito básico, pode ser demonstrado que é equivalente, do ponto de vista de complexidade, ao modelo da máquina RAM: qualquer problema que pode ser resolvido num modelo em tempo polinomial, pode também ser resolvido no outro em tempo polinomial.

A máquina é constituída por uma cabeça de leitura/escrita, que lê e escreve símbolos numa fita de comprimento infinito. Pode considerar-se que a fita está dividida em células, sendo possível escrever apenas um símbolo em cada célula. A Figura A.3 apresenta um modelo de uma máquina de Turing.

A máquina possui um conjunto finito de estados, citando-se, entre outros, um estado inicial q_0, e estados de paragem q_S (aceitação) e q_N (não-aceitação). Dependendo do estado de paragem, a resposta ao problema de decisão colocado é afirmativa ou negativa.

A operação da máquina traduz-se numa sequência de passos, de acordo com um programa, que é definido por:

(a) um conjunto finito de símbolos de um alfabeto,

(b) um conjunto finito de estados

(c) uma tabela de transições, T:

A tabela de transições decide a acção a tomar em função do estado em que se encontra a máquina e do símbolo acabado de ler. Para cada um dos possíveis pares, é necessário definir:

(a) o símbolo a ser deixado na posição que acabou de ser lida, q'; a máquina pode escrever um símbolo novo ou deixar o existente;

	b			1		
q_0	1	q_1	+1	1	q_0	+1
q_1	b	q_2	-1	1	q_1	+1
q_2				b	stop	

Tabela A.4: Tabela de Transições

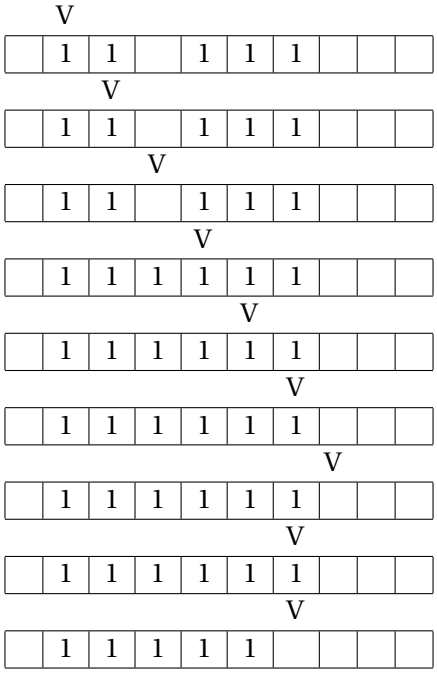


Tabela A.5: Operação de uma Máquina de Turing

(b) o estado para o qual a máquina passa antes de se movimentar para uma nova posição, s' ;

(c) o movimento a efectuar, d, ou a decisão de paragem; todos os movimentos são de apenas 1 posição, ou para a direita (+1), ou para a esquerda (-1).

A tabela de transições pode ser expressa como uma função:

$$T(q,s) := (q',s',d)$$

Exemplo A.7. Máquina de Turing que procede à adição de dois números representados na base unária, isolados por separadores (espaços). A máquina remove o espaço que separa os dois números do input, apaga o último símbolo do segundo número e pára. □

A.4 Problemas Polinomiais

Pertencem à classe de problemas polinomiais os problemas cuja solução pode ser obtida num número de oerações que pode ser expresso como uma função polinomial do comprimento de uma codificação dos dados do problema usando um modelo de computação determinística. Para

provar a inclusão na classe, basta exibir um algoritmo que responda ao problema de decisão em tempo polinomial. Dentro da classe de problemas polinomiais, é usual identificar os problemas de acordo com a ordem de grandeza do algoritmo de solução, por exemplo, por $O(n^3)$. O grau do polinómio é suficiente para especificar a complexidade do problema.

Exemplo A.8. Considere o problema de decidir se um grafo possui, ou não, um emparelhamento perfeito. Um algoritmo baseado no teorema de Hall não serviria para provar que o problema do emparelhamento perfeito é polinomial.

O algoritmo teria de verificar, para cada subconjunto de vértices de V_1, se a vizinhança do subconjunto era de cardinal maior que o do subconjunto em consideração. Um algoritmo baseado neste processo não seria polinomial, porque seria necessário verificar um número de subconjuntos que é exponencial em função do número de vértices do grafo.

Para provar a inclusão do problema de decisão na classe P, é necessário exibir um algoritmo polinomial, no pior caso, como, por exemplo, o algoritmo de emparelhamento máximo apresentado. Este algoritmo permite dar uma resposta ao problema de decisão de uma forma eficiente. □

A.4.1 Equivalência de Problemas de Decisão e de Optimização

Um problema de optimização combinatória é definido por $\{\max cx : x \in X\}$, sendo X o conjunto dos pontos válidos. A este problema de optimização pode ser associado um problema de decisão que é: dados um conjunto X e uma constante K, existe algum $x \in X$ tal que $cx \geq K$? Quando um problema de decisão é polinomial, o problema de optimização que lhe está associado é também polinomial. As duas versões do problema são equivalentes do ponto de vista de complexidade.

Para provar este resultado, é necessário mostrar que a solução de um problema de optimização pode ser reduzida à solução de uma sequência com um número polinomial de problemas de decisão. Sendo o número de problemas da sequência polinomial e sabendo–se que o esforço envolvido na solução de cada problema de decisão é polinomial, o esforço envolvido na solução do problema de optimização será também polinomial, dadas as propriedades da multiplicação de funções polinomiais.

Vamos, em primeiro lugar, mostrar que o número de problemas de sequência é polinomial. Seja M um limite superior para o valor do óptimo do problema de optimização. É importante reconhecer que este valor deve poder ser limitado superiormente, não podendo ser exponencialmente grande em função da dimensão da instância. Se isso acontecesse, ter–se–ia uma instância do problema de decisão que não poderia ela própria ser resolvida em tempo polinomial.

Utilizando a técnica da pesquisa binária, número inteiro entre 1 e M pode ser determinado fazendo, no máximo, $\lceil \log_2 M \rceil$ questões da forma $x \geq K$?

Exemplo A.9. Qualquer número no intervalo [1,17] pode ser determinado fazendo $\lceil \log_2 17 \rceil = 5$ questões (o mesmo valor é suficiente para o intervalo [1,32]), conforme é ilustrado no seguinte Exemplo:

intervalo	questão	resposta
1...32	$x \geq 16$?	não
1...15	$x \geq 8$?	não
1...7	$x \geq 4$?	sim
4...7	$x \geq 6$?	sim
6...7	$x \geq 7$?	sim

□

Para a redução ser válida, deve ser também possível responder em tempo polinomial, quer a resposta seja "sim"ou "não". Embora para problemas polinomiais estas questões sejam equivalentes, há classes de problemas para as quais a complexidade de uma questão e da questão complementar podem ser diferentes.

A.5 Problemas NP

A classe NP é a classe de problemas de decisão que podem ser resolvidos em tempo polinomial usando uma máquina não-determinística. As iniciais significam (N)ão-determinístico em tempo (P)olinomial.

Um certificado é uma informação (solução do problema) que, conjuntamente com a definição da instância, tornam possível verificar que a resposta a um problema de decisão é "sim". Na escala de complexidade, os problemas de NP são problemas que têm um certificado conciso, *i.e.*, de comprimento polinomial, que pode ser verificado em tempo polinomial.

As dificuldades não residem nas características do certificado, que é algo conciso, nem na sua verificação, que pode ser feita de uma forma eficiente, mas na própria obtenção de um certificado que sirva para responder afirmativamente à questão colocada.

Por outro lado, saliente–se que, se fosse possível verificar, em simultâneo, todos os candidatos possíveis, que podem ser em número exponencial, o problema de decisão seria resolvido em tempo polinomial.

Exemplo A.10. O problema do circuito Hamiltoniano em grafos não–orientados pertence à classe NP. Uma instância deste problema é definida, por exemplo, pelo conjunto de arcos que constituem o grafo não-orientado. Um certificado para este problema consiste, por exemplo, num conjunto de arcos que constituam um circuito. Este certificado é conciso, porque o seu comprimento é polinomial em função dos dados da instância.

Dado um certificado, é possível verificar em tempo polinomial que se trata de um circuito Hamiltoniano. Para o efeito, é necessário confirmar que os arcos do certificado pertencem ao grafo e que formam um único circuito que passa por todos os vértices. Claramente, ambas estas operações podem ser executadas em tempo polinomial. □

Os únicos algoritmos conhecidos para a solução deste problema podem, no pior caso, envolver um número exponencial de operações. O facto de não ter sido ainda descoberto nenhum algoritmo polinomial para a obtenção de um certificado, usando computação determinística, não significa que o número de operações necessárias para esse efeito seja exponencial. Sendo o certificado conciso, nada obsta a que venha ainda a ser descoberto um algoritmo que possa construir esse certificado num tempo polinomial.

Por outro lado, se se conseguisse provar que existiam problemas de NP que requeressem necessariamente um número exponencial de operações, isso significaria que existiam problemas pertencentes a NP e não a P, sendo, portanto, P um conjunto próprio de NP, *i.e.*, diferente do próprio NP.

Os problemas de P pertencem também à classe NP, porque também possuem um certificado conciso (que, aliás, se pode construir em tempo polinomial) que é possível verificar em tempo polinomial. Conforme foi afirmado, nada obsta a que venha ainda a ser descoberto um algoritmo eficiente para a solução de todos os problemas de NP. É, no entanto, convicção generalizada que tais algoritmos não existem. A conjectura de P=NP permanece em aberto.

A.5.1 Máquinas de Turing Não-determinísticas

Uma máquina de Turing não-determinística tem a possibilidade de, em cada passo, criar várias réplicas de si própria, que prosseguem independentemente a execução do algoritmo, podendo, cada uma delas, por sua vez, replicar-se. O algoritmo termina quando uma das cópias encontra um certificado, que verifica em tempo polinomial, que permite responder afirmativamente à questão de decisão colocada.

Enquanto no modelo determinístico existe apenas uma acção para cada par (símbolo × estado), na tabela de transições do modelo não–determinístico pode haver k acções alternativas:

$$T(q,s) := \{(q'_1, s'_1, d_1), (q'_2, s'_2, d_2), ..., (q'_k, s'_k, d_k)\}$$

A este processo, pode ser associada uma árvore em que os ramos pendurados em cada nó correspondem às diferentes acções alternativas. Para a solução poder ser dada em tempo polinomial, é necessário que o nível de profundidade da árvore seja polinomial e que seja possível avaliar, em tempo polinomial, usando computação determinística, se a solução correspondente a cada vértice da árvore responde afirmativamente ao problema de decisão colocado.

Para todas as instâncias de um problema de decisão, diz-se que uma máquina de Turing não–determinística resolve o problema de decisão em tempo polinomial se:

(a) a resposta for afirmativa para a instância, existe uma sequência de escolhas que permite construir um certificado conciso que a máquina de Turing verifica em tempo polinomial, atingindo o estado de aceitação e parando.

(b) a resposta for negativa para a instância, não existe nenhum certificado que leve a máquina a parar no estado de aceitação.

No caso (b), a máquina pode parar no estado de não-aceitação ou, então, não parar.

A.5.2 Problemas NP-completos

Dentro da classe de problemas NP, existe uma classe de problemas que podem ser considerados como os mais difíceis da classe NP. São designados por problemas NP-completos. Esses problemas constituem uma classe de equivalência em termos de complexidade, porque existem técnicas que permitem reduzir os problemas entre si em tempo polinomial. Como corolário deste facto, se se vier a descobrir um algoritmo polinomial para a solução de um qualquer problema NP–completo, todos os restantes problemas da classe poderão ser também resolvidos em tempo polinomial. Os dois conjuntos são ilustrados na Figura A.2.

Problema da Satisfação Booleana

O primeiro problema que se provou pertencer à classe de problemas NP–completos foi o problema da satisfação booleana. Considere um conjunto de variáveis $\{x_1, x_2, \ldots, x_n\}$ e designe-se o complemento (negação) de cada variável x_i por $\bar{x}_i$. Estas variáveis, designadas por literais, podem tomar o valor lógico verdadeiro ou falso. O símbolo $\wedge$ designa a operação lógica *e* e o símbolo $\vee$ designa a operação *ou*. Qualquer expressão Booleana pode ser reduzida à forma normal conjuntiva, ou seja, à conjunção de um número finito de disjunções, onde cada literal aparece apenas uma vez.

$$(x_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2) \wedge (x_2 \vee x_3)$$

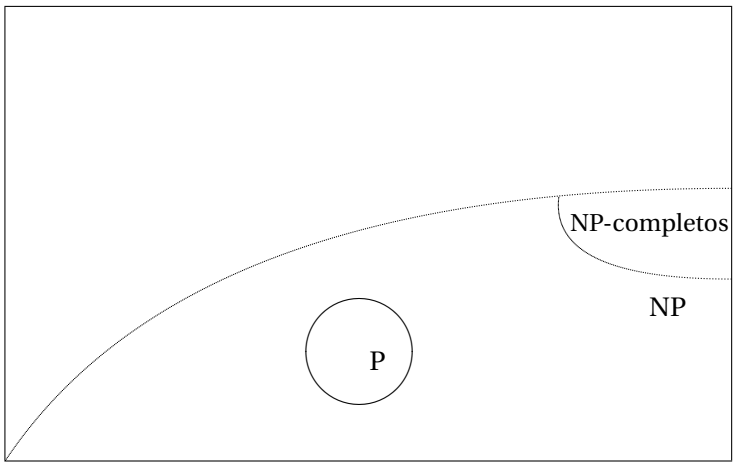


Figura A.2: Complexidade de Problemas de Decisão

Dado um conjunto de literais e uma conjunção de cláusulas, existirá uma afectação de valores lógicos aos literais que tornem a expressão verdadeira? Se existir, diz-se que a expressão pode ser satisfeita.

Para a expressão ser verdadeira, é necessário que todas as cláusulas sejam verdadeiras; para isso, pelo menos um literal de cada disjunção deve ser verdadeiro. Para a expressão apresentada, os valores lógicos x_2 =verdadeiro e $x_1 = x_3$ = falso satisfazem a expressão.

Teorema 7 (Cook)**.** *O problema da satisfação Booleana (SAT) é NP–completo.*

Não iremos aqui fazer a prova deste Teorema, mas apenas apresentar as ideias principais. Antes de mais o problema da satisfação booleana pertence claramente à classe NP. Dado um certificado, é possível verificar em tempo polinomial que a expressão booleana é satisfeita.

Para provar que o problema da satisfação booleana pertence à classe de problemas NP–completos, é necessário mostrar que este problema é dos mais difíceis da classe NP. Para provar este Teorema, Cook mostrou que a operação de qualquer máquina de Turing não–determinística pode ser modelada como uma expressão booleana.

Quando a instância possui a propriedade questionada no problema de decisão, existe, pelo menos, uma sequência de escolhas, de comprimento polinomial, que levam a máquina de Turing desde o estado inicial até ao estado de aceitação. Se o número de passos efectuado for polinomial, o comprimento de fita utilizado também o será.

As sequências de escolhas que levam a máquina até ao estado de aceitação constituem um subconjunto das escolhas possíveis. O ponto fulcral do Teorema de Cook é a demonstração de que o conjunto de todas as sequências possíveis de operação pode ser modelado como uma expressão booleana. A expressão booleana é satisfeita quando corresponde a uma qualquer computação válida que conduza ao estado de aceitação. O número de variáveis e o número de cláusulas usados na expressão booleana é polinomial em função dos dados do problema.

Deste modo, qualquer problema da classe NP pode ser modelado como um problema de satisfação booleana. □

Uma forma de resolver o problema da satisfação booleana seria enumerar todas as hipóteses possíveis, verificando, para cada uma delas, se a expressão é satisfeita. Embora a verificação possa

ser feita de uma forma eficiente, há número exponencial de possibilidades a testar. Neste caso, porque as variáveis podem tomar dois valores diferentes, há um total de 2^n possibilidades diferentes.

Outros problemas NP-completos

Com base em técnicas de reduções que serão abordadas na próxima Secção, Karp demonstrou que muitos problemas da área da optimização combinatória eram também NP-completos. Da lista elaborada, citam-se os seguintes problemas de decisão:

Circuito Hamiltoniano

Dado um grafo $G = (V, A)$, existe um circuito Hamiltoniano?

Programação Inteira 0-1

Dada uma matriz A e um vector inteiro b, existirá um vector x, de elementos 0-1, tal que $Ax = b$?

Saco de Mochila Binário

Dado um conjunto de inteiros $w_j,\ j = 1, 2, \ldots, n,$ e um valor K, existe um subconjunto $S : \sum_{j \in S} w_j = K$?

Clique

Um clique de ordem k é um subgrafo completo S com k vértices. Dado um grafo $G = (V, A)$ e um $k \leq \mid V \mid$, existe uma clique de ordem superior a k?

Vértices Independentes

Um subconjuntos de vértices, S, é independente, se nenhum par de vértices for adjacente, *i.e.*, se não existir nenhum arco entre os dois vértices.

Dado um grafo $G = (V, A)$ e um $k \leq \mid V \mid$, existe um conjunto de vértices independentes de cardinal $\geq k$?

Cobertura do Grafo

Um subconjunto de vértices, S, é uma cobertura do grafo, se todos os arcos forem incidentes em, pelo menos, um dos vértices de S.

Dado um grafo $G = (V, A)$ e um $k \leq \mid V \mid$, existe uma cobertura com menos de k vértices?

Exemplo A.11. A Figura A.3 apresenta exemplos de uma clique, um conjunto de vértices independentes e uma cobertura do grafo. Conforme será analisado, os três problemas de decisão são equivalentes. □

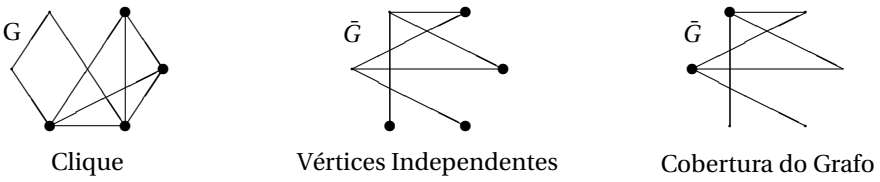


Figura A.3: Problemas NP-completos definidos em grafos

A.5.3 Redução de Problemas

Para provar que um problema pertence à classe de problemas NP–completos, podem ser utilizadas técnicas de reduções entre problemas em tempo polinomial. Um problema A diz-se redutível a B se qualquer instância do problema A puder ser transformada em tempo polinomial numa instância do problema B e se as respostas aos problemas de decisão forem equivalentes nas duas instâncias. Claramente, a complexidade de A é menor ou igual à soma da complexidade de transformar A em B e da complexidade de B.

A classe de problema NP-completos congrega os problemas mais difíceis de NP. Se o problema A reconhecidamente pertencer à classe de problemas NP-completos, o problema B em análise é identificado como tendo um certificado conciso, não sendo a sua solução mais fácil que a de um problema que já pertence à classe.

Assim, para provar a inclusão na classe de problemas NP–completos é necessário mostrar que:

(a) o problema possui um certificado conciso e que pertence à classe NP;

(b) existe um problema NP–completo que pode ser transformado no problema em análise em tempo polinomial.

Redução de Satisfação Booleana para Clique

O problema da clique pertence a NP, porque possui um certificado conciso que é possível verificar em tempo polinomial: dado um conjunto de vértices com o cardinal desejado, é necessário verificar que existe um arco entre cada par de vértices da clique. Além disso, um problema da satisfação booleana pode ser transformado em tempo polinomial num problema de Clique.

Para uma dada expressão Booleana, construa-se um grafo $G = (V, A)$, sendo $V = \{(\alpha, i) : \alpha$ é um literal da cláusula $i\}$ e $A = \{((\alpha, i), (\beta, j)) : i \neq j \ , \ \alpha \neq \tilde{\beta}\}$. O grafo terá um vértice para cada literal existente em cada cláusula. Existirá um arco entre cada par de vértices se os vértices corresponderem a cláusulas diferentes e o literal existente numa das cláusulas não for a negação do literal do outro vértice.

Exemplo A.12. Para a expressão $(x_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2) \wedge (x_2 \vee x_3)$, o grafo correspondente é o indicado na Figura A.4. Existe uma clique de dimensão 3, se a expressão booleana for satisfeita. □

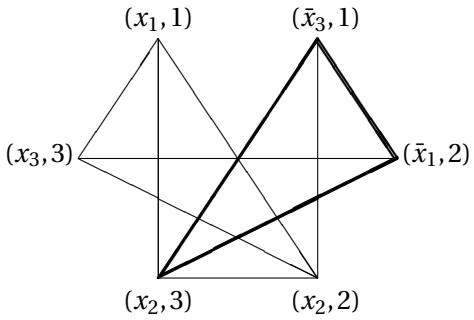


Figura A.4: Tranformação de Satisfação Booleana para Clique

Redução de Satisfação Booleana para Programação Inteira

O problema de programação inteira pertence a NP, porque possui um certificado conciso. Este facto não será aqui provado.

O problema da satisfação Booleana pode ser formulado como um problema de programação inteira, com variáveis x_j que tomam os valores 0-1. O valor 1 corresponde ao valor lógico verdadeiro e o valor 0 ao valor lógico falso. Cada cláusula é representada por uma restrição, em que os literais $\bar{x}_j$ são substituídos por $(1 - x_j)$. A operação lógico $\vee$ é substituída pela adição aritmética. Assim, a restrição da forma

$$\sum_{j:x_j} x_j + \sum_{j:\bar{x}_j} (1 - x_j) \geq 1$$

garante que, pelo menos, um dos literais assegura que a cláusula é verdadeira. Da mesma forma, o respeito de todas as restrições assegura a satisfação da expressão global.

Exemplo A.13. Considere o conjunto de cláusulas lógicas apresentadas no Exemplo anterior:

$$(x_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2) \wedge (x_2 \vee x_3)$$

A formulação de programação inteira com variáveis booleanas correspondente é:

$$\begin{array}{ll} x_1 + (1 - x_3) \geq 1 & x_1 - x_3 \geq 0 \\ (1 - x_1) + x_2 \geq 1 & -x_1 + x_2 \geq 0 \\ x_2 + x_3 \geq 1 & x_2 + x_3 \geq 1 \\ x_1,\ x_2,\ x_3 \text{ booleanos} & x_1,\ x_2,\ x_3 \text{ booleanos} \end{array}$$

□

Estas restrições definem um poliedro, que pode conter ou não pontos inteiros. Em caso afirmativo, a expressão Booleana é satisfeita. É importante realçar a diferença entre o poliedro inteiro definido pelas restrições e a respectiva relaxação linear, que resulta de não ser necessário observar as restrições de integralidade. O poliedro linear pode não ser vazio, *i.e.*, existe, pelo menos, um ponto que obedece a todas as restrições, como, por exemplo, $x_1 = x_2 = x_3 = \frac{1}{2}$. Este facto não garante que exista um ponto inteiro, se for necessário cumprir as restrições de integralidade.

A.5.4 Equivalência de Problemas

Se a redução polinomial funcionar nos dois sentidos, os problemas têm a mesma complexidade. Demonstrando a equivalência entre o problema em análise e um problema que reconhecidamente pertença à classe de problemas NP-completos, pode–se provar a sua inclusão na mesma classe.

Equivalência de Clique e de Vértices Independentes

Teorema 8. *Dado um grafo $G = (V, A)$ e um subconjunto S de V, as seguintes afirmações são equivalentes:*

(i) S é uma clique do grafo G;

(ii) S é um conjunto de vértices independentes de $\bar{G}$, o complemento de G.

Prova: Se S for uma clique de G, existe um arco entre cada par de vértices de S. No grafo complementar, $\bar{G}$, esses vértices são independentes. O mesmo acontece em sentido contrário. □

Neste caso, é fácil mostrar que a redução entre problemas funciona nos dois sentidos. Dada uma qualquer instância de um destes problemas, pode–se construir em tempo polinomial uma instância do outro problema, e sabe–se que se a resposta a um dos problemas de decisão for afirmativa, ela será também afirmativa para o outro problema. Aliás, o próprio certificado de solução pode também ser convertido em tempo polinomial.

Equivalência de Vértices Independentes e de Cobertura do Grafo

Dado um grafo $G = (V, A)$ e um subconjunto S de V, as seguintes afirmações são equivalentes:

(i) S é um conjunto de vértices independentes de G;

(ii) $V - S$ é uma cobertura do grafo G.

Prova: Nenhum arco do grafo pode ser incidente em dois vértices independentes. Se S é um conjunto de vértices independentes de G, todos os arcos do grafo devem ter, pelo menos, um terminal no conjunto $V - S$. O mesmo acontece em sentido contrário. □

Exemplo A.14. Conforme apresentado na Figura A.3, os vértices que constituem uma clique de G são vértices independentes em $\bar{G}$. Os restantes vértices do grafo $\bar{G}$ são uma cobertura do grafo. □

A.5.5 Problemas Pseudo-polinomiais

Dentro da classe de problemas NP-completos, existem problemas cuja complexidade é devida à sua própria estrutura e problemas cuja complexidade resulta da dimensão dos dados do problema. Estes últimos problemas, que são designados por problemas pseudo-polinomiais, podem ser resolvidos em tempo aceitável quando a dimensão dos dados é relativamente pequena.

A designação de pseudo-polinomial deriva do facto de ser possível resolver este problema em tempo polinomial em função da dimensão dos dados, quando os dados são codificados usando uma representação unária.

Exemplo A.15. O problema de mochila binário é um exemplo de um problema pseudo-polinomial. O comprimento necessário à codificação binária dos dados inteiros de um problema de mochila binário (supondo, sem perda de generalidade, que todos os itens $j, j = 1, \ldots, n$, têm um peso w_j inferior à capacidade da mochila W) é $\sum_j \lceil \log_2(w_j + 1) \rceil + n + \lceil \log_2(W + 1) \rceil + 1$. Este comprimento é da ordem de $n \log_2 W$.

Usando programação dinâmica, o problema do saco de mochila binário pode ser resolvido em $O(nW)$. O problema não é polinomial, porque nW não pode ser expresso como uma função

polinomial do comprimento da codificação dos dados do problema, que usa um espaço logarítmico.

No entanto, se considerarmos um valor fixo de W, o problema é polinomial, sendo a sua solução linear em função do número de itens. □

A.5.6 Problemas NP-completos Unários

Por outro lado, há problemas cuja dificuldade se mantém quaisquer que sejam os dados envolvidos. Mesmo limitando os valores dos dados do problema a um dado valor máximo, o problema permanece difícil. Estes problemas são designados por problemas fortemente NP-completos.

Representação aritmética - Nesta representação, considera-se que cada dado do problema ocupa apenas uma posição, quaisquer que sejam os números representados.

Um exemplo é o problema do caixeiro viajante cuja dificuldade intrínseca não é diminuída se for imposto um limite ao valor máximo que os custos podem tomar. Se isso acontecesse, poder-se-ia resolver o problema do circuito Hamiltoniano em tempo polinomial. De facto, o problema do circuito Hamiltoniano é um problema de decisão que pode ser modelado como um problema de caixeiro viajante cujos arcos têm todos um custo unitário. A existência de um circuito Hamiltoniano pode ser decidida em função do valor do óptimo do problema do caixeiro viajante. Se o valor do óptimo for finito, igual ao número de vértices do grafo, a solução correspondente é um circuito Hamiltoniano.

Sendo todos os dados unitários, a representação deste modelo requer um comprimento equivalente, quer a codificação seja feita numa base binária ou unária. Portanto, um algoritmo pseudo-polinomial resolveria o problema do circuito Hamiltoniano em tempo polinomial.

Estes problemas são também designados por NP-completos unários, significando que permanecem NP-completos mesmo sob uma representação unária, por contraposição com o caso geral de problemas NP-completos binários.

A.6 Problemas NP-difíceis

Existem problemas para os quais não foi estabelecido pertencerem à classe NP, por, por exemplo, ainda não ter sido estabelecido para eles um certificado que seja conciso. No entanto, se o problema constituir uma generalização de um problema pertencente à mesma classe ou se existir algum problema NP-completo que lhe seja redutível, então a sua solução não será mais fácil que a solução de qualquer problema NP-completo. Estes problemas designam-se por problemas NP-difíceis. Esta designação serve para classificar os problemas de optimização cuja formulação de decisão é um problema NP-completo. Um problema NP-difícil só poderá ser resolvido em tempo polinomial se P=NP.

A.7 Problemas de co-NP

Um problema pertence à classe co-NP se a decisão "não"puder ser tomada em tempo polinomial usando computação não-determinística. Exceptuando os problemas da classe P, não se conhece nenhum problema de co-NP que pertença à classe NP, porque não existem certificados concisos que permitam que eles sejam incluídos nesta classe.

Os problemas de $P \in NP \cap$ co-NP, porque é possível utilizar o algoritmo polinomial para construir um certificado conciso que permite verificar se a resposta é afirmativa ou negativa, sendo o

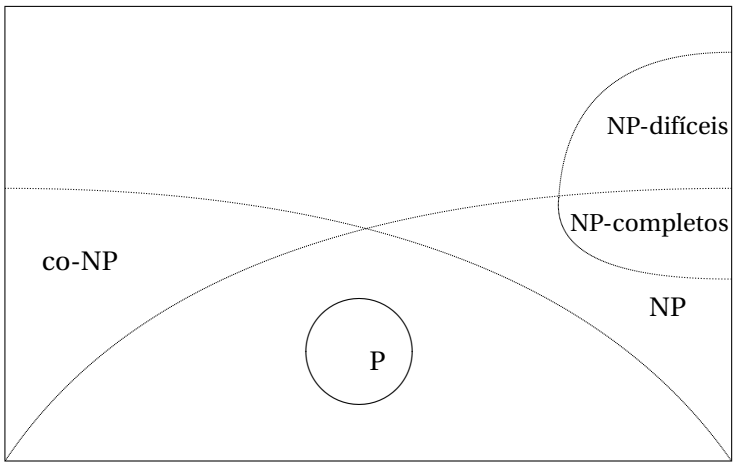


Figura A.5: Conjecturas sobre Classes de Complexidade

esforço envolvido nas duas decisões equivalente. Conjectura-se que estes são os únicos problemas que pertencem à intersecção de NP e de co-NP. A Figura A.5 ilustra a relação entre estas classes, incluindo também os problemas NP-difíceis.

As diferenças existentes entre os problemas que reconhecidamente pertencem a P e os problemas de NP são ilustradas pelos Exemplos apresentados de seguida.

Exemplo A.16. O problema do fluxo máximo pertence à classe de problemas polinomiais, bem como a correspondente versão de decisão: dada uma rede $R = (V, A, s, t)$ e um inteiro k, existe algum fluxo $\geq k$. Este problema pertence a NP, porque uma solução válida de fluxo $\geq k$ constitui um certificado conciso que é possível verificar em tempo polinomial.

Por outro lado, o problema de decisão complementar, *i.e.*, dada uma rede e um k, não existe nenhum fluxo $\geq k$?, é um problema cuja resposta afirmativa também pode ser verificada polinomialmente. O problema complementar também pertence a NP, porque possui um certificado conciso, que é dado por um corte. Um corte de capacidade inferior a k permite verificar que não pode existir nenhum fluxo de valor superior ao requerido. □

Exemplo A.17. O problema do circuito Hamiltoniano pode ser formalizado como um problema de decisão: dado um grafo $G = (V, A)$, será que o grafo possui um circuito Hamiltoniano? Este problema pertence a NP, porque um circuito Hamiltoniano válido

Por outro lado, o problema complementar, *i.e.*, dado um grafo, será que o grafo não possui um circuito Hamiltoniano?, é um problema para o qual não se conhece nenhum certificado conciso que permita verificar uma resposta afirmativa a este problema em tempo polinomial. Portanto, a inclusão deste problema na classe NP ainda não foi provada.

Uma possível forma de verificar que um grafo não é Hamiltoniano seria enumerar todos os circuitos do grafo e verificar que nenhum era Hamiltoniano. A questão é que este certificado não seria conciso, porque o número de circuitos existentes num grafo é, normalmente, uma função exponencial do número de vértices. □

Boas Caracterizações

Na generalidade, os teoremas caracterizam uma propriedade em termos da existência ou não de uma estrutura, apresentando condições necessárias e suficientes à existência da propriedade.

Existem teoremas que caracterizam a existência de uma propriedade à custa da existência de uma outra propriedade. Por outro lado, há teoremas, que Edmonds designou por "boas caracterizações", que caracterizam a existência de uma propriedade à custa da não–existência de uma outra propriedade.

São exemplos de "boas caracterizações"o teorema de Hall e o teorema do fluxo máximo / corte mínimo. Dada uma qualquer instância de um destes problemas, é possível verificar de uma forma eficiente se a instância possui, ou não, a propriedade, a partir da análise de um certificado conciso. A designação de "boa caracterização "resulta deste facto.

Todos os problemas polinomiais têm "boas"caracterizações, porque é possível apresentar os certificados que provam a existência ou a não–existência da propriedade. Por outro lado, para a generalidade dos problemas para os quais foram estabelecidas boas caracterizações, vieram também a ser descobertos algoritmos polinomiais, conjecturando-se que todos os problemas com "boas"caracterizações podem ser resolvidos em tempo polinomial.

A.8 Notas

O conceito de "bom"algoritmo e de "boa"caracterização foram introduzidos por Edmonds, que pode ser considerado um percursor da teoria da complexidade. O modo como surgiram alguns desses conceitos, bem como o seu enquadramento nas ideias da época, é descrito em Edmonds [**?**]

> Jack Edmonds. A glimpse of heaven. Em History of Mathematical Programming, A Collection of Personal Reminiscences. CWI, North Holland, 1991.

A prova de que o problema da satisfação booleana pertence à classe de problemas NP–completos aparece em Cook [2].

[]

> S.A. Cook. The complexity of theorem-proving procedures. In Proceedings of the 3rd Annual ACM Symposium on Theory of Computing Machinery, pages 151–158, 1972.

Utilizando técnicas de redução entre problemas, é possível provar que pertencem também a esta classe cerca de duas dezenas de problemas:

> R.M. Karp. Reducibility among combinatorial problems. In Complexity of Computer Computations, pages 85–103. Plenum Press, 1972.

O primeiro livro publicado sobre teoria da complexidade incluía uma lista de mais de 300 problemas NP–completos e NP–difíceis de campos extremamente variados:

> M.R. Garey and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP–Completeness. W.H.Freeman and Co., San Francisco, 1979.

Esta última referência inclui uma prova do Teorema de Cook. Outra referência sobre este assunto é:

C.H. Papadimitriou and K. Steiglitz. Combinatorial Optimization: Algorithms and Complexity. Prentice–Hall, Inc., Englewood Cliffs, New Jersey, 1982.

A equivalência entre modelos de computação, nomeadamente a máquina RAM e a máquina de Turing, é apresentada por

A. Aho, J. Hopcroft, e J. Ullman. The Design and Analysis of Computer Algorithms. Addison–Wesley Publishing Co., Reading, Massachussetts, 1974.

Anexo B

Definições e Conceitos Elementares de Grafos

Um grafo é um modelo ou uma representação de um sistema real constituído por vértices e por arcos ou arestas. As aplicações e a investigação feitas no domínio dos grafos deram origem a um corpo muito vasto de resultados e técnicas, que tem encontrado inúmeras aplicações em campos muito diversos, como, por exemplo, a matemática, a engenharia e as ciências sociais.

Neste Capítulo, iremos apenas apresentar as definições básicas e os conceitos necessários usados na apresentação dos problemas. São também apresentados alguns conceitos elementares que serão utilizados numa avaliação do desempenho dos algoritmos de solução.

B.1 Introdução

Um grafo é constituído por um conjunto de vértices (ou nós) e por um conjunto de arcos ou arestas, que são linhas ou ramos unindo os vértices. Dependendo das necessidades, iremos considerar dois tipos distintos de grafos, grafos orientados e grafos não–orientados, consoante os arcos / arestas do grafo têm ou não um sentido associado.

B.2 Grafos Não-Orientados

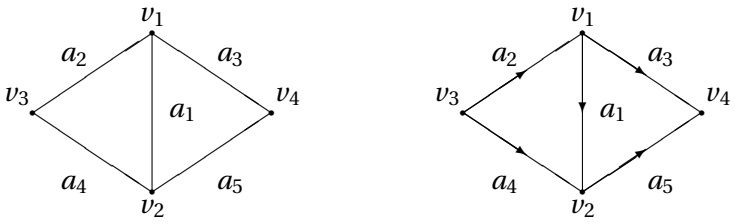
Um grafo G é um par (V, A) definido por um conjunto de vértices $V = \{v_1, v_2, \ldots, v_n\}$ e por um conjunto de arestas $A = \{a_1, a_2, \ldots, a_m\}$. Cada arco a_k liga um par de vértices distintos, designados por extremos ou terminais do arco. Representa-se por $a_k = (v_i, v_j) : v_i, v_j \in V \text{ e } v_i \neq v_j$. Não iremos considerar grafos com anéis, *i.e.*, arcos cuja origem e cujo destino são o mesmo vértice.

Os arcos que têm o vértice v_j como extremo são designados por arcos incidentes no vértice v_j.

A ordem de um grafo é o número de vértices do grafo, que é igual ao cardinal do conjunto V, designado por $| V |$.

Exemplo B.1. Considere o grafo não–orientado de ordem 4 apresentado na Figura B.1. O conjunto de vértices $V = \{v_1, v_2, v_3, v_4\}$ e o conjunto de arestas $A = \{a_1, a_2, a_3, a_4, a_5\}$, ou, numa representação alternativa, $A = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_2, v_3), (v_2, v_4)\}$. A arestas incidentes no vértice v_1 são a_1, a_2 e a_3. □

Um grafo que possua mais que um arco não-orientado entre o mesmo par de vértices, designa–se por multigrafo.



Grafo não–orientado Grafo orientado

Figura B.1: Grafos

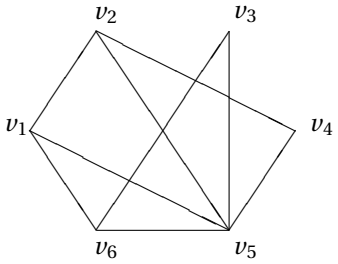


Figura B.2: Grafo Não-orientado

Adjacência

Dois vértices unidos por um arco (ou aresta) designam-se por adjacentes. O conjunto dos vértices adjacentes ao vértice x_i designa–se por $\Gamma(x_i)$. A mesma definição pode ser estendida a um conjunto de vértices S. O conjunto de vértices adjacentes a S designa–se por $\Gamma(S)$.

Exemplo B.2. Considere o grafo apresentado na Figura B.2. Os vértices adjacentes ao vértice v_3 são os vértices v_5 e v_6, *i.e.*, $\Gamma(v_3) = \{v_5, v_6\}$. Por outro lado, se considerarmos o conjunto $S = \{v_3, v_4\}, \Gamma(\{v_3, v_4\}) = \{v_2, v_5, v_6\}$. □

Grau de um vértice

O grau de um vértice v_j, designado por d_{v_j}, é igual ao número de arcos incidentes em v_j. Se todos os vértices tiverem o mesmo grau d, o grafo designa-se por grafo regular de grau d. A Figura B.3 apresenta vários grafos regulares de grau 3.

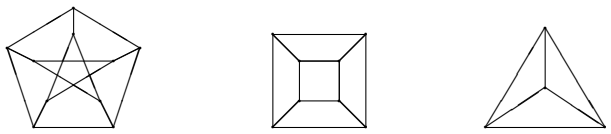


Figura B.3: Grafos regulares de grau 3

Subgrafos

Um subgrafo G' representa parte de um grafo G, designando–se por próprio se $G' \neq G$.

Subgrafo induzido por um conjunto de vértices

Dado um grafo $G = (V, A)$ e um conjunto de vértices $V' \subseteq V$, o subgrafo induzido por V' é o grafo $G' = (V', A')$ constituído por V' e por todos os arcos $A' \subseteq A$ com ambos os terminais incidentes em vértices de V'.

Subgrafo induzido por um conjunto de arcos

Dado um grafo $G = (V, A)$ e um conjunto de arcos $A' \subseteq A$, o subgrafo induzido por A' é o grafo $G' = (V, A')$ constituído pelos vértices do grafo original e pelos arcos A'.

Grafos Completos

Um grafo é completo se existir um arco entre cada par de vértices. Um grafo completo, ou clique, de ordem n é designado por K_n.

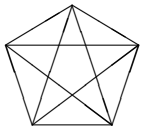


Figura B.4: Grafo completo K_5

Complemento de um Grafo

O complemento de um grafo G é um grafo, designado por $\bar{G}$, constituído pelo mesmo conjunto de vértices, mas tendo arcos apenas entre os vértices que não são adjacentes em G.

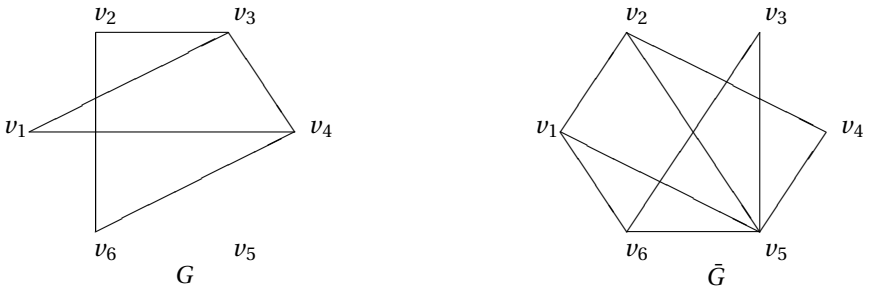


Figura B.5: Um grafo e o seu complemento

Caminhos e Circuitos

Um caminho (ou caminho simples) é uma sequência de vértices distintos $a, b, c, \ldots, q$, conjuntamente com os arcos ligando os vértices consecutivos $(a, b), (b, c), \ldots, (p, q)$. Se forem permitidas repetições de nós, os caminhos são não–simples.

Um circuito é um caminho fechado, em que o vértice de início e de fim coincidem. Um circuito é Hamiltoniano se percorrer todos os vértices do grafo. A Figura B.6 apresenta dois grafos: para o primeiro, mostra–se um circuito hamiltoniano; o segundo não possui nenhum circuito hamiltoniano.

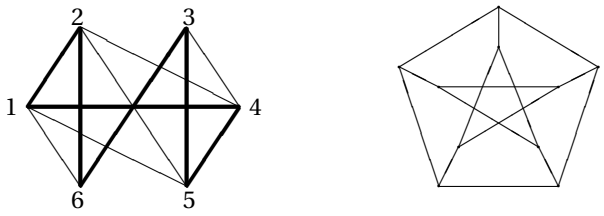


Figura B.6: Um Circuito Hamiltoniano e um Grafo sem Circuitos Hamiltonianos

Grafos ligados e Componentes

Um grafo é ligado se existir um caminho entre qualquer par de nós. Se existir um par de nós entre os quais não exista nenhum caminho, então o grafo pode ser dividido em, pelo menos, duas partes, cada uma delas ligada. Cada uma dessas partes é designada por componente do grafo.

Grafos Bipartidos

Um grafo é bipartido se o conjunto de vértices V puder ser dividido em dois conjuntos disjuntos, V_1 e V_2, $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$, de tal modo que todos os arcos (i, j) tenham origem num vértice $i \in V_1$ e destino num vértice $j \in V_2$. Os grafos bipartidos serão designados por $G = (V_1, V_2, A)$.

Grafos Bipartidos Completos

Um grafo bipartido é completo se existir um arco entre cada vértice de V_1 e cada vértice de V_2. Um grafo bipartido completo é designado por $K_{m,n}$, sendo $m =\mid V_1 \mid$ e $n =\mid V_2 \mid$ o número de vértices de V_1 e V_2, respectivamente.

Teorema 9. Um grafo é bipartido se e só se não tiver nenhum circuito de comprimento ímpar.

Prova: ($\Rightarrow$) Vamos supor que existe um circuito de comprimento ímpar. Neste circuito, há vértices de V_1 e V_2 e arcos unindo os vértices. Para o circuito ser de comprimento ímpar, deve existir, pelo menos, um arco entre dois vértices de V_1, ou entre dois vértices de V_2, o que contraria a definição.

($\Leftarrow$) Sem perda de generalidade, vamos supor que o grafo é constituído apenas por um componente; caso contrário, poder–se–ia repetir o argumento para cada componente. A prova é baseada num algoritmo de coloração dos vértices. Se todos os circuitos forem de comprimento par, então

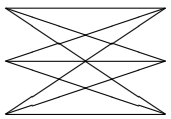


Figura B.7: Grafo bipartido completo $K_{3,3}$

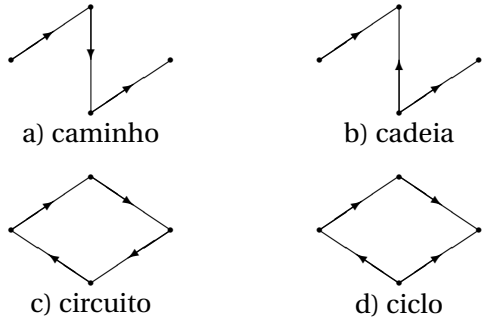


Figura B.8: Definições

é possível colorir os vértices pertencentes a cada circuito, alternadamente, de cor vermelha e de cor azul. Se dividirmos os vértices em dois conjuntos, segundo a cor, encontramos uma bipartição dos vértices. □

B.3 Grafos Orientados

Quando existe uma orientação associada aos arcos, os grafos designam–se por grafos orientados ou digrafos, a partir de *directed graphs*. Neste caso, cada arco (i, j) tem a sua origem no vértice i e o seu destino no vértice j.

Caminhos, Cadeias, Circuitos e Ciclos

Um caminho (ou caminho simples) é uma sequência de vértices distintos $a, b, c, \ldots, q$, conjuntamente com os arcos ligando os vértices consecutivos $(a, b), (b, c), \ldots, (p, q)$. Se forem permitidas repetições de nós, os caminhos são não–simples.

Uma cadeia é semelhante a um caminho, excepto o facto de ser permitido percorrer alguns dos arcos em sentido inverso. Um circuito é um caminho fechado e um ciclo é uma cadeia fechada. A Figura B.8 ilustra estes conceitos.

B.4 Representação de Grafos

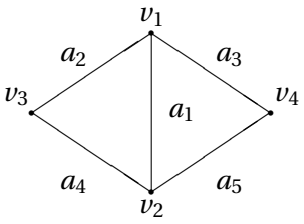
B.4.1 Matriz de Incidência nó–Arco

Grafos Não–Orientados

Seja $G = (V, A)$ um grafo com n vértices e com m arcos. A matriz de incidência nó–arco, B, é uma matriz com n linhas e com m colunas. Cada coluna, que representa um arco do grafo, tem dois elementos iguais a 1, nas posições correspondentes aos vértices em que o arco é incidente, *i.e.*,

$$b_{ij} = \begin{cases} 1 & \text{, se } (v_i, v_j) \in A \\ 0 & \text{, caso contrário} \end{cases}$$

Na Figura B.9, apresenta-se um grafo não-orientado e a respectiva matriz de incidência nó-arco.

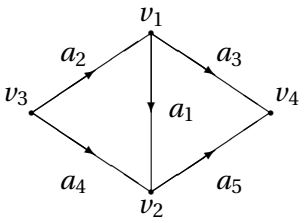


Grafo não–orientado

$$A = \begin{array}{c} \begin{array}{ccccc} a_1 & a_2 & a_3 & a_4 & a_5 \end{array} \\ \left(\begin{array}{ccccc} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{array} \right) \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \end{array}$$

Matriz de incidência nó-arco

Figura B.9: Matriz de incidência de um grafo não–orientado



Grafo orientado

$$A = \begin{array}{c} \begin{array}{ccccc} a_1 & a_2 & a_3 & a_4 & a_5 \end{array} \\ \left(\begin{array}{ccccc} 1 & -1 & 1 & 0 & 0 \\ -1 & 0 & 0 & -1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & -1 \end{array} \right) \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \end{array}$$

Matriz de incidência nó-arco

Figura B.10: Matriz de incidência de um grafo orientado

Grafos Orientados

Seja $G = (V, A)$ um grafo com n vértices e com m arcos. A matriz de incidência nó–arco, B, é uma matriz com n linhas e com m colunas. Cada coluna, que representa um arco do grafo, tem dois elementos diferentes de 0, um elemento +1 na linha correspondente ao vértice de origem e um elemento −1 na linha correspondente ao vértice de destino. Na Figura B.10, apresenta-se um grafo orientado e a respectiva matriz de incidência nó-arco.

B.4.2 Matriz de Adjacências

Seja $G = (V, A)$ um grafo não-orientado com n vértices. A matriz de adjacência de um grafo, B, é uma matriz quadrada de ordem n. Esta matriz tem elementos iguais a 1 nas posições correspondentes a vértices adjacentes, *i.e.*,

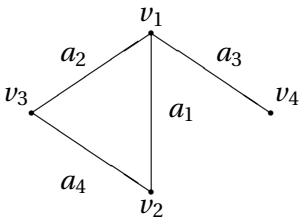
$$b_{ij} = \begin{cases} 1 & \text{, se } (v_i, v_j) \in A \\ 0 & \text{, caso contrário} \end{cases}$$

Na Figura B.11, apresenta-se um grafo não-orientado e a respectiva matriz de adjacências.

B.4.3 Estrela de Sucessores

A matriz de incidência nó–arco é uma matriz que tem um número de linhas igual ao número de vértices do grafo e um número de colunas igual ao número de arcos do grafo. Para grafos de grande dimensão, o espaço requerido para guardar toda a informação pode ser proibitivamente grande.

Uma forma mais económica de representação consiste em guardar, para cada vértice, apenas os vértices que lhe são sucessores. Um vértice v_j é sucessor do vértice v_i, se existir um arco orientado (v_i, v_j). Esta representação é designada por estrela de sucessores.



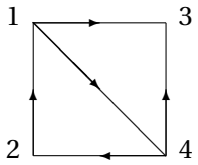
Grafo não–orientado

$$A = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \left(\begin{array}{cccc} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{array} \right) \end{array} \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array}$$

Matriz de adjacências

Figura B.11: Matriz de adjacências de um grafo não–orientado

Exemplo B.3. Considere o seguinte grafo orientado, constituído por 4 vértices e 5 arcos.



Na representação da estrela de sucessores, são utilizados dois vectores. No primeiro vector, $SUC()$, guardam-se os vértices nos quais as cabeças dos arcos são incidentes. Este vector tem 5 posições, tantas quantas os arcos do grafo.

arco	1	2	3	4	5
	1	1	2	4	4
$SUC()$	3	4	1	2	3

Nesta representação, os arcos que partem de um determinado vértice são guardados em posições contíguas. Este facto permite as as origens dos arcos não sejam guardadas explicitamente. Em vez disso, guardam–se apenas as posições em que começam os arcos que partem de cada vértice. Essa informação é colocada num segundo vector, $A()$, cujas posições apontam para as posições do vector $SUC()$ onde começam os arcos correspondentes a cada vértice. Para manter a consistência, é acrescentada uma posição a este vector, que informa qual a última posição do vector $SUC()$ que está preenchida. Assim, este vector tem 5 posições, igual ao número de vértices mais um.

vértice	1	2	3	4	–
$A()$	1	3	4	4	6

□

É de salientar a diferença entre o espaço requerido para as duas representações aqui apresentadas. Para um grafo com n vértices e m arcos, na representação da matriz de incidência nó–arco são necessárias $m \times n$ posições, enquanto na representação da estrela de sucessores são necessárias $m + (n+1)$ posições.

Notas

A teoria de grafos abrange múltiplos aspectos como, entre outros, a coloração de vértices, a planaridade e a enumeração de grafos. Estes aspectos são abordados nas seguintes referências:

Claude Berge. Graphs and Hypergraphs. North-Holland, 1973.

BIBLIOGRAFIA

Bibliografia

[1] R. Ahuja, T. Magnanti, and J. Orlin. Network Flows: Theory, Algorithms and Applications. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[2] S. A. Cook. The complexity of theorem-proving procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.

[3] G. Desaulniers, J. Desrosiers, Y. Dumas, S. Marc, B. Rioux, M. Solomon, and F. Soumis. Crew pairing at Air France. European Journal of Operational Research, 97(2):245 – 259, 1997.