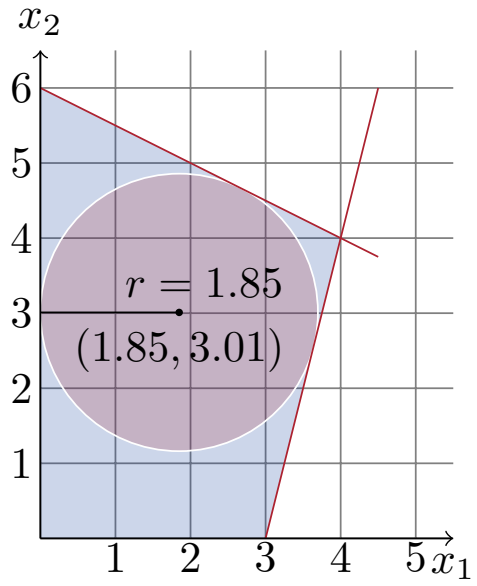
INF05010 – Otimização combinatória



Notas de aula

Marcus Ritt
marcus.ritt@inf.ufrgs.br

10 de fevereiro de 2022

Universidade Federal do Rio Grande do Sul
Instituto de Informática
Departamento de Informática Teórica

Versão 11907 do 2022-02-10, compilada em 10 de fevereiro de 2022.

Na parte I, as notas de aula seguem o livro "Linear programming: Foundations and extensions" de Robert J. Vanderbei, Universidade Princeton, disponível em `http://www.princeton.edu/~rvdb/LPbook`. Agradeço contribuições de Luciana Buriol e Alysson M. Costa às primeiras versões dessas notas.

Fonte das imagens:
George Dantzig (18): INFORMS, Jean Baptiste Joseph Fourier (18): Wikipedia, Xadrez (104): Wikipedia, Mauricio G. C. Resende (172): Página pessoal, Fred Glover (175): Página pessoal, Pierre Hansen (179): Página pessoal, Pablo Moscato (192): Página pessoal.

Conteúdo

Parte I.

Programação linear

1. Introdução

Introdução

> If one would take statistics about which mathematical problem is using up most of the computer time in the world, then ... the answer would probably be linear programming. (Laszlo Lovasz)

1.1. Exemplo

Exemplo 1.1 (No Ildo)
Antes da aula visito o Ildo[1] para tomar um café e comer um Croissant. Ele me conta: "Estou especializado em Croissants e Strudels. Tenho um lucro de 20 centavos por Croissant e 50 centavos por Strudel. Diariamente até 80 clientes compram um Croissant e até 60 um Strudel." Mas infelizmente, o Ildo apenas disponibiliza de 150 ovos e 6 kg de açúcar por dia. Entre outros ingredientes, preciso um ovo e 50g de açúcar para cada Croissant e um ovo e meio e 50g de açúcar para cada Strudel. "Agora, professor, quantas Croissants e Strudels devo produzir para obter o maior lucro?"

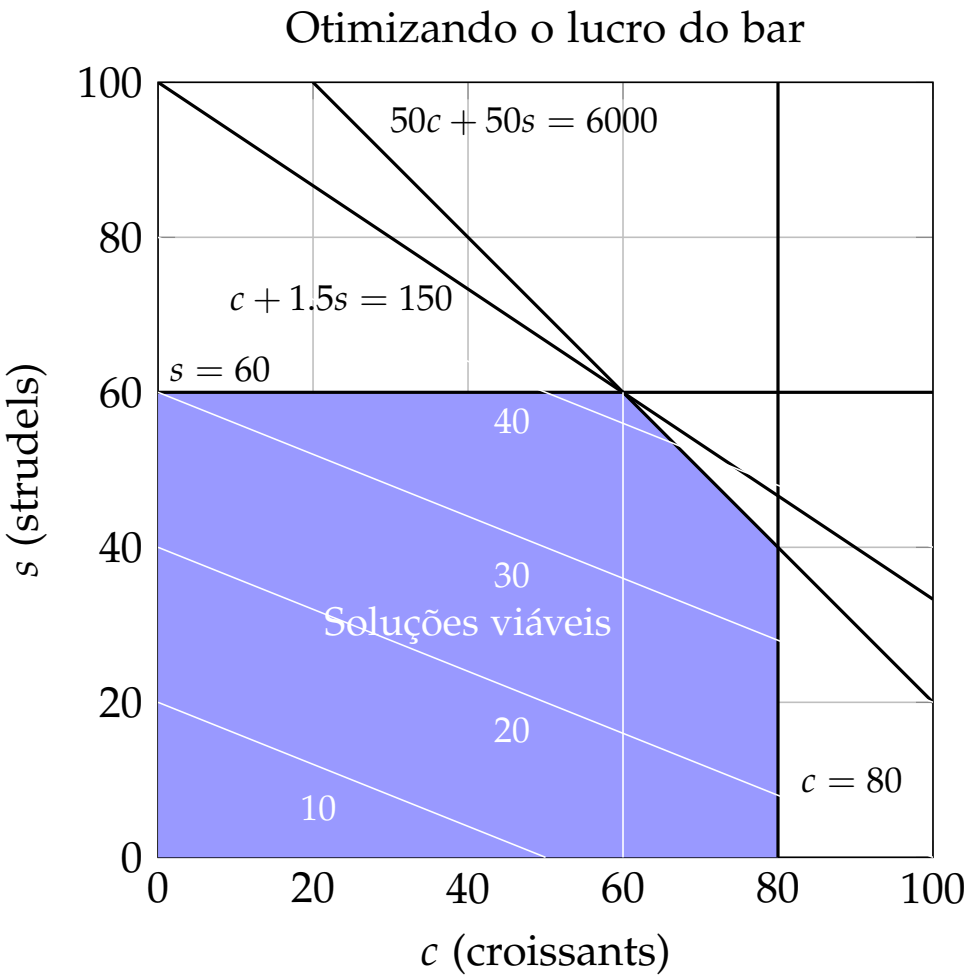
Sejam c o número de Croissants e s o número de Strudels. O lucro do Ildo em Reais é $0.2c + 0.5s$. Seria ótimo produzir todos 80 Croissants e 60 Strudels, mas uma conta simples mostra que não temos ovos e açúcar suficiente. Para produzir os Croissants e Strudels precisamos $c + 1.5s$ ovos e $50c + 50s$g de açúcar que não podem ultrapassar 150 ovos e 6000g. Com a condição óbvia que $c \geq 0$ e $s \geq 0$ chegamos no seguinte problema de otimização:

$$\begin{aligned} \textbf{maximiza} \quad & 0.2c + 0.5s \\ \textbf{sujeito a} \quad & c + 1.5s \leq 150, \\ & 50c + 50s \leq 6000, \\ & c \leq 80, \\ & s \leq 60, \\ & c, s \geq 0. \end{aligned} \tag{1.1}$$

[1] Uma lancheria que existia no Instituto de Informática até 2012.

Como resolver esse problema? Com duas variáveis podemos visualizar a situação num grafo com c no eixo x e s no eixo y

No Ildo



que nesse caso permite resolver o problema graficamente. Desenhando diversos *conjunto de nível* (ingl. *level set*) com valor da função objetivo 10, 20, 30, 40 é fácil observar que o lucro máximo encontra-se no ponto $c = s = 60$, e possui um valor de 42 reais.

$\Diamond$

Definição 1.1 (Conjunto de nível (ingl. level set))
Para uma função $f(x)$, $x \in \mathbb{R}^n$ a conjunto de nível $c \in \mathbb{R}$ é $L_c(f) = \{x \mid f(x) = c\}$.

A forma geral de um *problema de otimização* (ou de *programação matemática*) é

$$\begin{aligned} \textbf{opt} \quad & f(x) \\ \textbf{sujeito a} \quad & x \in V, \end{aligned}$$

com

- um *objetivo* **opt** $\in \{\max, \min\}$,
- uma *função objetivo* (ou função critério) $f : V \to \mathbb{R}$,
- um conjunto de *soluções viáveis* (ou soluções candidatas) V.

Falamos de um *problema de otimização combinatória*, caso V é discreto.
Nessa generalidade um problema de otimização é difícil ou impossível de resolver. O exemplo 1.1 é um problema de *otimização linear* (ou *programação linear*):

- as variáveis de decisão são reais: $x_1, \ldots, x_n \in \mathbb{R}$
- a função de otimização é linear em $x_1, \ldots, x_n$:

$$f(x_1, \ldots, x_n) = c_1 x_1 + \cdots + c_n x_n \tag{1.2}$$

- as soluções viáveis são definidas implicitamente por m *restrições lineares*

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \bowtie_1 b_1, \tag{1.3}$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \bowtie_2 b_2, \tag{1.4}$$
$$\cdots \tag{1.5}$$
$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \bowtie_m b_m, \tag{1.6}$$

com $\bowtie_i \in \{\leq, =, \geq\}$.

Exemplo 1.2 (O problema da dieta (Dantzig))
Suponha que temos uma tabela de nutrientes de diferentes tipos de alimentos. Sabendo o valor diário de referência (VDR) de cada nutriente (quantidade de nutriente que deve ser ingerido) e o preço de cada unidade de

alimento, qual a dieta ótima, i.e. a dieta de menor custo que contém pelo menos o valor diário de referência?
Com m nutrientes e n alimentos, seja a_{ij} a quantidade do nutriente i no alimento j (em g/g), r_i o valor diário de referência do nutriente i (em g) e c_j o preço do alimento j (em R\$/g). Queremos saber as quantidades x_j de cada alimento (em g) que

$$\begin{aligned}
\textbf{minimiza} \quad & c_1x_1 + \cdots + c_nx_n && (1.7)\\
\textbf{sujeito a} \quad & a_{11}x_1 + \cdots + a_{1n}x_n \geq r_1, && (1.8)\\
& \cdots \\
& a_{m1}x_1 + \cdots + a_{mn}x_n \geq r_m, && (1.9)\\
& x_1, \ldots, x_n \geq 0. && (1.10)
\end{aligned}$$

◊

Exemplo 1.3 (Problema de transporte (Hitchcock))
Uma empresa agrária tem m depósitos, cada um com um estoque de a_i, $i \in [m]$ toneladas de milho. Ela quer encaminhar b_j, $j \in [n]$ toneladas de milho para n clientes diferentes. O transporte de uma tonelada do depósito i para cliente j custa R\$ c_{ij}. Qual seria o esquema de transporte de menor custo?
Para formular o problema linearmente, podemos introduzir variáveis x_{ij} que representam o peso dos produtos encaminhados do depósito i ao cliente j, e queremos resolver

$$\begin{aligned}
\textbf{minimiza} \quad & \sum_{i\in[m], j\in[n]} c_{ij}x_{ij} && (1.11)\\
\textbf{sujeito a} \quad & \sum_{j\in[n]} x_{ij} \leq a_i, \qquad \text{para todo fornecedor } i \in [m], && (1.12)\\
& \sum_{i\in[m]} x_{ij} = b_j, \qquad \text{para todo cliente } j \in [n], && (1.13)\\
& x_{ij} \geq 0, \qquad \text{para todo fornecedor } i \in [m] \text{ e cliente } j \in [n].
\end{aligned}$$

Concretamente, suponha que temos a situação da Figura 1.1. A figura mostra as toneladas disponíveis de cada fornecedor, a demanda (em toneladas) de cada cliente e as distâncias (em km) entre eles. O transporte custa R\$ 1000 por km e tonelada. Observe que um transporte do fornecedor 1 para cliente 3 e fornecedor 3 para cliente 1 não é possível. Nós usaremos uma distância grande de 100 km nesses casos (uma outra possibilidade é usar restrições

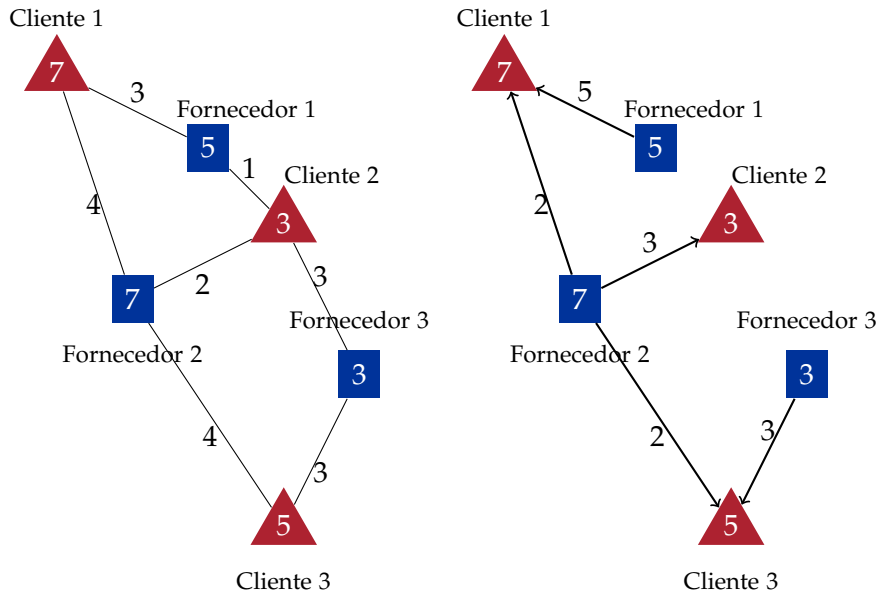


Figura 1.1.: Esquerda: Instância do problema de transporte. Direita: Solução ótima correspondente.

$x_{13} = x_{31} = 0$ ou remover as variáveis x_{13} e x_{31} do modelo).

$$
\begin{aligned}
\textbf{minimiza} \quad & 3x_{11} + x_{12} + 100x_{13} + 4x_{21} + 2x_{22} \\
& + 4x_{23} + 100x_{31} + 3x_{32} + 3x_{33}, \\
\textbf{sujeito a} \quad & x_{11} + x_{12} + x_{13} \leq 5, \\
& x_{21} + x_{22} + x_{23} \leq 7, \\
& x_{31} + x_{32} + x_{33} \leq 3, \\
& x_{11} + x_{21} + x_{31} = 7, \\
& x_{12} + x_{22} + x_{32} = 3, \\
& x_{13} + x_{23} + x_{33} = 5, \\
& x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23}, x_{31}, x_{32}, x_{33} \geq 0.
\end{aligned}
$$

Qual seria a solução ótima? A Figura 1.1 (direita) mostra o número ótimo de toneladas transportadas. O custo mínimo é 46 (em R$ 1000). ◊

Podemos simplificar a descrição de um programa linear usando notação matricial. Com $A := (a_{ij}) \in \mathbb{R}^{m \times n}$, $b := (b_i) \in \mathbb{R}^m$, $c := (c_i) \in \mathbb{R}^n$ e

$x = (x_i) \in \mathbb{R}^n$ o problema 1.2-1.6), pode ser escrito de forma

$$\begin{array}{rl} \textbf{opt} & c^t x \\ \textbf{sujeito a} & a_i x \bowtie_i b_i, \qquad i \in [m] \end{array}$$

(Denotamos com a_i a i-ésima linha e como a^j a j-ésima coluna da matriz A.) Em caso todas restrições usam a mesma relação $\leq$, $\geq$ ou $=$ podemos escrever

$$\begin{array}{rl} \textbf{opt} & c^t x \\ \textbf{sujeito a} & Ax \leq b, \end{array} \qquad \begin{array}{rl} \textbf{opt} & c^t x \\ \textbf{sujeito a} & Ax \geq b, \end{array} \quad \text{ou} \quad \begin{array}{rl} \textbf{opt} & c^t x \\ \textbf{sujeito a} & Ax = b. \end{array}$$

Exemplo 1.4 (Problema do Ildo em forma matricial)
O problema 1.1 em forma matricial é

$$\begin{array}{rl} \textbf{maximiza} & (0.2\ 0.5)(c\ s)^t \\ \textbf{sujeito a} & \begin{pmatrix} 1 & 1.5 \\ 50 & 50 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} c \\ s \end{pmatrix} \leq \begin{pmatrix} 150 \\ 6000 \\ 80 \\ 60 \end{pmatrix}, \\ & (c\ s) \geq 0. \end{array}$$

$\Diamond$

Observação 1.1 ("Programar" linearmente)
Como explicado na seção histórica 1.4, o termo "programação" em "programação linear" se refere a "agendamento" ou "planejamento". Porém, formular programas lineares é uma atividade muito similar à programação de computadores. Um programa linear consiste de declarações de variáveis, constantes, uma função objetivo e uma série de restrições. Podemos escrever um programa linear de forma mais "computacional" para enfatizar a similaridade com programas. No caso do problema de Hitchcock 1.3, por exemplo, podemos escrever

```
var   x_ij , i ∈ [m], j ∈ [n]          { declaração variáveis }
const a_i , i ∈ [m]                    { estoques }
const b_j , j ∈ [n]                    { demandas }
max   Σ_{i∈[m],j∈[n]} c_ij x_ij
st    Σ_{j∈[n]} x_ij ≤ a_i , i ∈ [m]   { limite estoque }
st    Σ_{i∈[m]} x_ij = b_j , j ∈ [n]   { satisfação demanda }
```

Podemos ainda, igual a programação, introduzir nomes para funções lineares para facilitar a formulação. Por exemplo enviado$(i) = \sum_{j\in[n]} x_{ij}$ é a quantidade total enviada pelo i-ésimo fornecedor. Similarmente, podemos escrever recebido$(j) = \sum_{i\in[n]} x_{ij}$ para a quantidade total recebida pelo j-ésimo cliente. Com isso nosso "programa" linear fica

```
var    x_ij , i ∈ [m], j ∈ [n]          { declaração variáveis }
const  a_i , i ∈ [m]                    { estoques }
const  b_j , j ∈ [n]                    { demandas }
const  c_ij , i ∈ [m], j ∈ [n]  { custos }
function enviado(i) = Σ_{j∈[n]} x_ij
function recebido(j) = Σ_{i∈[m]} x_ij
max    Σ_{i∈[m],j∈[n]} c_ij x_ij
st     enviado(i) ≤ a_i , i ∈ [m]  { limite estoque }
st     recebido(j) = b_j , j ∈ [n]  { satisfação demanda }
```

Vamos conhecer linguagens reais para especificar programas lineares no parte prático. Um exemplo é Julia/JuMP explicado no appéndice B. A nossa especificação acima pode ser vista como "pseudo-código" de uma linguagem atual como Julia/JuMP. ♢

1.2. Formas normais

Conversões

É possível converter

- um problema de minimização para um problema de maximização

$$\min c^t x \iff -\max -c^t x$$

  (o sinal $-$ em frente do max é uma lembrança que temos que negar a solução depois.)

- uma restrição "$\geq$" para uma restrição "$\leq$"

$$a_i x \geq b_i \iff -a_i x \leq -b_i$$

- uma igualdade para desigualdades

$$a_i x = b_i \iff a_i x \leq b_i \wedge a_i x \geq b_i$$

Conversões

- uma desigualdade para uma igualdade

$$a_i x \leq b \iff a_i x + x_{n+1} = b_i \wedge x_{n+1} \geq 0$$
$$a_i x \geq b \iff a_i x - x_{n+1} = b_i \wedge x_{n+1} \geq 0$$

  usando uma nova *variável de folga ou excesso* x_{n+1} (inglês: slack and surplus variables).

- uma variável x_i sem restrições para duas não-negativas

$$x_i^+ \geq 0 \wedge x_i^- \geq 0$$

  substituindo x_i por $x_i^+ - x_i^-$.

Essas transformações permitem descrever cada problema linear em uma *forma padrão*.

Forma padrão

$$\begin{aligned} \textbf{maximiza} \quad & c^t x \\ \textbf{sujeito a} \quad & Ax \leq b, \\ & x \geq 0. \end{aligned}$$

As restrições $x \geq 0$ se chamam *triviais*.

Exemplo 1.5
Dado o problema

$$\begin{aligned} \textbf{minimiza} \quad & 3x_1 - 5x_2 + x_3 \\ \textbf{sujeito a} \quad & x_1 - x_2 - x_3 \geq 0, \\ & 5x_1 + 3x_2 + x_3 \leq 200, \\ & 2x_1 + 8x_2 + 2x_3 \leq 500, \\ & x_1, x_2 \geq 0. \end{aligned}$$

vamos substituir "**minimiza**" por "**maximiza**", converter a primeira desigualdade de $\geq$ para $\leq$ e introduzir $x_3 = x_3^+ - x_3^-$ com duas variáveis positivas x_3^+ e x_3^- para obter a forma padrão

$$\begin{aligned} \textbf{maximiza} \quad & -3x_1 + 5x_2 - x_3^+ + x_3^- \\ \textbf{sujeito a} \quad & -x_1 + x_2 + x_3^+ - x_3^- \leq 0, \\ & 5x_1 + 3x_2 + x_3^+ - x_3^- \leq 200, \\ & 2x_1 + 8x_2 + 2x_3^+ - 2x_3^- \leq 500, \\ & x_1, x_2, x_3^+, x_3^- \geq 0. \end{aligned}$$

Em notação matricial temos

$$c = \begin{pmatrix} -3 \\ 5 \\ -1 \\ 1 \end{pmatrix}; \quad b = \begin{pmatrix} 0 \\ 200 \\ 500 \end{pmatrix}; \quad A = \begin{pmatrix} -1 & 1 & 1 & -1 \\ 5 & 3 & 1 & -1 \\ 2 & 8 & 2 & -2 \end{pmatrix}.$$

$\Diamond$

Definição 1.2 (Soluções viáveis, inviáveis e ótimas)
Para um programa linear P em forma normal, um vetor $x \in \mathbb{R}^n$ é uma *solução viável*, caso $Ax \leq b$ e $x \geq 0$. P é *viável* caso existe alguma solução viável, caso contrário P é *inviável*. Um vetor $x^* \in \mathbb{R}^n$ é uma *solução ótima* caso $c^t x^* = \max\{c^t x \mid Ax \leq b, x \geq 0\}$.

Definição 1.3 (Programas ilimitados)
Uma programa linear em forma normal é *ilimitado* caso existe um $v \in \mathbb{R}$ tal que para todo $w \geq v$ existe uma solução viável x com $c^t x \geq w$.

1.3. Solução por busca exaustiva

Uma observação importante na solução de um programa linear é que a solução ótima, caso exista, somente ocorra na borda de região das soluções viáveis (compara com a figura na página 8). Mais específico a solução ótima ocorre num vértice (ou ponto extremo) dessa região, definido pela interseção de n restrições linearmente independentes. Isso justifica tratar a programação linear como problema de otimização combinatória, porque temos um número finito de $\binom{m}{n}$ candidatos para a solução ótima. Procurando o melhor entre todos candidatos nos também fornece um algoritmo (muito ineficiente) para encontrar uma solução ótima de um programa linear, caso exista.

Definição 1.4
Um conjunto $C \subseteq \mathbb{R}^n$ é *convexo*, caso para todo par de pontos $x, y \in C$ a sua *combinação convexa* $\lambda x + (1-\lambda)y$ para $\lambda \in [0,1]$ também pertence a C.

Proposição 1.1
A região de soluções viáveis $V = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ definido por um programa linear é convexa.

Prova. Sejam $x, y \in V$. Então

$$A(\lambda x + (1-\lambda)y) = \lambda Ax + (1-\lambda)Ay \leq \lambda b + (1-\lambda)b = b.$$

■

Definição 1.5
Um ponto $x \in C$ de uma região $C \subseteq \mathbb{R}^n$ é um *vértice* ou *ponto extremo*, caso não existe um $y \neq 0$ tal que $x + y \in C$ e $x - y \in C$.

Proposição 1.2
Caso existe uma única solução ótima de $\max\{c^t x \mid x \in V\}$ ela é um vértice de V.

Prova. Supõe que a solução ótima x^* não é um vértice de V. Então existe um y tal que $x + y \in V$ e $x - y \in V$. Por x^* ser a única solução ótima temos $c^t(x^* + y) < c^t x^*$ e $c^t(x^* - y) < c^t x^*$, i.e., $c^t y < 0$ e $-c^t y < 0$, uma contradição. ■

Proposição 1.3
Um vértice de $V = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ é a interseção de n restrições linearmente independentes.

Prova. Para um vértice $v \in V$, seja A_v a matriz formado das linhas a_i de A tal que $a_i v = b_i$, e b_v os lados direitos correspondentes.
Seja $v \in V$ a interseção de n restrições linearmente independentes, i.e. $\text{posto}(A_v) = n$. Supõe v não é um vértice. Logo existe um y tal que $x + y, x - y \in V$ que satisfazem $A_v(x + y) \leq b_v$ e $A_v(x - y) \leq b_v$. Como $A_v x = b_v$ obtemos $A_v y \leq 0$ e $-A_v y \leq 0$, i.e. $A_v y = 0$, uma contradição com $\text{posto}(A_v) = n$.
Agora seja $v \in V$ um vértice e supõe $\text{posto}(A_v) < n$, i.e. existe um y tal que $A_v y = 0$. Para as linhas a_i em A com $a_i v < b_i$ existe um $\delta > 0$ tal que

$$a_i(v + \delta y) \leq b_i \text{ e } a_i(v - \delta y) \leq b_i$$

e logo

$$A(v + \delta y) \leq b \text{ e } A(v - \delta y) \leq b,$$

porque $A_v y = 0$, em contradição com o fato que v é um vértice. ■

Proposição 1.4
Caso existem múltiplas soluções ótimas de $\max\{c^t x \mid x \in V\}$ e V é limitado, um vértice de V é uma solução ótima.

Prova. Por indução sobre $n - \text{posto}(A_v)$. Caso $n - \text{posto}(A_v) = 0$, v é um vértice pela proposição (1.3). Para $n - \text{posto}(A_v) > 0$ existe um y com $A_v y = 0$. Seja $\mu = \max\{t \mid v + ty \in V\}$. O valor μ existe porque V é limitado (e compacto). Como $a_i(v + \mu y) \leq b_i$ para cada linha i temos que

$$\mu = \min\{(b_i - a_i v)/a_i y \mid a_i y > 0\} \qquad (+)$$

Seja i^* o índice da linha que satisfaz (+) com igualdade. Define $v' = v + \mu y$. Temos $A_v v' = A_v v + \mu A_v y = A_v v = b_v$, logo $A_{v'}$ contém as linhas de A_v e pelo menos a linha a_{i^*} a mais. Ainda, como $A_v y = 0$ mas $a_{i^*} y \neq 0$ temos que $\text{posto}(A_{v'}) > \text{posto}(A_v)$. Logo, pela hipótese da indução, existe um vértice que é uma solução ótima. ■

Observação 1.2
Caso existem multiplas soluções ótimas de $\max\{c^t x \mid x \in V\}$, mas V não é limitado, é possível que não existe um vértice ótimo. Um exemplo é o sistema $\max\{x_1 \mid (x_1, x_2) \in \mathbb{R}^2, 0 \leq x_1 \leq 1\}$. ◊

Aplicando a proposição 1.4 obtemos um algoritmo simples para resolver sistemas lineares, que enumera todos vértices e retorna o vértice de maior valor.

Algoritmo 1.1 (Solução de programas linear por exaustão)
Entrada Programa linear $\max\{c^t x \mid Ax \leq b, x \in \mathbb{R}^n_+\}$.

```
x* := null
for todas (m n) seleções de n restrições lin. indep.
  determine a interseção x das n restrições
  if Ax ≤ b e c^t x ≥ c^t x* then
    x* := x
  end if
end for
if x* ≠ null then
```

```
  return "Solução ótima é x* ou sistema ilimitado"
else
  return "Não possui solução ou não possui vértice"
end if
```

1.4. Notas históricas

História da programação linear

- Jean Baptiste Joseph Fourier (1826): Método de resolver um sistema de desigualdades (eliminação de Fourier-Motzkin) (Williams 1986).
- Leonid Kantorovich (1939): Programação linear.
- George Bernard Dantzig (1948): Método Simplex.
- John von Neumann: Dualidade.
- Leonid Khachiyan (1979): Método de ellipsoides.
- Narendra Karmarkar (1984): Métodos de pontos interiores.

Figura 1.2.: Jean Baptiste Joseph Fourier (*1768, +1830)

Pesquisa operacional, otimização e "programação"

- "The discipline of applying advanced analytical methods to help make better decisions" (INFORMS)
- O nome foi criado durante a segunda guerra mundial, para métodos científicos de análise e predição de problemas logísticos.
- Hoje se aplica para técnicas que ajudam tomar decisões sobre a execução e coordenação de operações em organizações.
- Problemas da pesquisa operacional são problemas de otimização.
- "Programação" não é "Programação"
  - Não se refere à computação: a noção significa "planejamento" ou "agendamento".

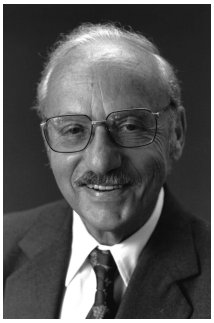


Figura 1.3.: George Bernard Dantzig (*1914, +2005)

Técnicas da pesquisa operacional

- Em geral: Técnicas algorítmicas conhecidas como
  - Modelagem matemática, e.g. equações, igualdades, desigualdades, modelos probabilísticos.
  - Algoritmos gulosos, randômicos, ...; programação dinâmica, linear, convexa, ...
  - Heurísticas e algoritmos de aproximação.
- Algumas dessas técnicas se aplicam para muitos problemas e por isso são mais comuns:
  - Exemplo: Programação linear.

1.5. Exercícios

(Soluções a partir da página 211.)

Exercício 1.1
Na definição da programação linear permitimos restrições lineares da forma

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \bowtie_i b_i$$

com $\bowtie_i \in \{\leq, =, \geq\}$. Por que não permitimos $\bowtie_i \in \{<, >\}$ também? Discute.

Exercício 1.2
Procura a tabela nutricional de algum restaurante e resolve o problema da dieta (exemplo 1.2).

Exercício 1.3
Um investidor pode vender ações de suas duas empresas na bolsa de valores, mas está sujeito a um limite de 10.000 operações diárias (vendas por dia). Na cotação atual, as ações da empresa A valorizaram-se 10% e agora cada uma vale R$ 22. Já a empresa B teve valorização de 2% e cada ação vale R$ 51. Sabendo-se que o investidor possui 6.000 ações da Empresa A e 7.000 da empresa B, maximize seu lucro na BOVESPA e diga qual o lucro obtido.

Exercício 1.4
Dona Maria adora ver seus netinhos Marcos, Renato e Vinicius bem alimentados. Sempre na hora de cozinhar ela leva em conta o quanto eles gostam

de cada prato para fazê-los comer o máximo possível. Marcos gosta da lasanha e comeria 3 pratos dela após um prato de sopa; Renato prefere lanches, e comeria 5 hambúrgueres, ignorando a sopa; Vinicius gosta muita da massa a bolonhesa, e comeria 2 pratos após tomar dois pratos de sopa. Para fazer a sopa, são necessários entre outros ingredientes, 70 gramas de queijo por prato e 30 gramas de carne. Para cada prato de lasanha, 200 gramas de queijo, e 100 gramas de carne. Para cada hambúrguer são necessários 100 gramas de carne, e 100 gramas de queijo. Para cada prato de massa a bolonhesa são necessários 100 gramas de carne e 30 gramas de queijo (ralado para por sobre a massa). Seus netos vieram visitá-la de surpresa, e tendo ela somente 800 gramas de carne e 1000 gramas de queijo em casa, como ela poderia fazê-los comer o maior número de pratos, garantindo que cada um deles comerá pelo menos dois pratos, e usando somente os ingredientes que ela possui?

Exercício 1.5
A empresa "Luz para o mundo" produz dois tipos de lampadas, cada um com partes metálicos e partes eléctricos. A gerencia quer saber com quantas unidades produzidas por tipo o lucro é maximizado. A produção de uma unidade de produto 1, precisa uma unidade de partes metálicos e duas unidades de componentes eléctricos. A produção de uma unidade de produto 2, precisa três unidades de partes metálicos e duas unidades de componentes eléctricos. A empresa tem um estoque de 200 unidades de partes metálicos e 300 unidades de componentes eléctricos. Cada unidade de produto um tem um lucro de R$ 1 e cada unidade de produto 2, até um limite de 60 unidades, um lucro de R$ 2. (Cada unidade acima de 60 no caso do produto 2 não rende nada.)

Exercício 1.6
A empresa "Janela jóia" com três empregados produz dois tipos de janelas: com molduras de madeira e com molduras de alumínio. Eles têm um lucro de 60 R$ para toda janela de madeira e 30R$ para toda janela de alumínio. João produz as molduras de madeira. Ele consegue produzir até seis molduras por dia. Sylvana é responsável pelas molduras de alumínio, e ela consegue produzir até quatro por dia. Ricardo corta o vidro e é capaz de produzir até 48 m^2 por dia. Uma janela de madeira precisa 6 m^2 de vidro, e uma de alumínio 8 m^2. A empresa quer maximizar o seu lucro.
Formule como programa linear.

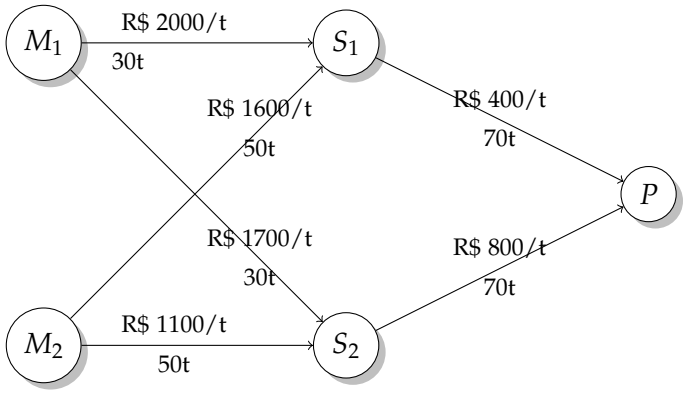


Figura 1.4.: Rede de distribuição de uma empresa de aço.

Exercício 1.7
Uma empresa de aço tem uma rede de distribuição conforme Figura 1.4. Duas minas P_1 e P_2 produzem 40t e 60t de mineral de ferro, respectivamente, que são distribuídos para dois estoques intermediários S_1 e S_2. A planta de produção P tem uma demanda dem 100t de mineral de ferro. A vias de transporte tem limites de toneladas de mineral de ferro que podem ser transportadas e custos de transporte por tonelada de mineral de ferra (veja figura). A direção da empresa quer determinar a transportação que minimiza os custos. Formule o problema como programa linear.

Exercício 1.8
Um importador de Whisky tem as seguintes restrições de importação

- no máximo 2000 garrafas de *Johnny Ballantine* por 70 R$ cada uma,
- no máximo 2500 garrafas de *Old Gargantua* por 50 R$ cada uma,
- no máximo 1200 garrafas de *Misty Deluxe* por 40 R$ cada uma.

Dos Whiskies importados ele produz três misturas A, B, C, que ele vende por 68 R$, 57 R$ e 45 R$, respectivamente. As misturas são

- A: no mínimo 60% Johnny Ballantine, no máximo 20% Misty Deluxe,
- B: no mínimo 15% Johnny Ballantine, no máximo 60% Misty Deluxe,
- C: no máximo 50% Misty Deluxe.

Quais seriam as misturas ótimas, e quantas garrafas de cada mistura devem ser produzidas para maximizar o lucro? Formule como programa linear.

Observações:

- Use como variáveis o número de garrafas $x_{m,i}$ da marca m usadas na mistura i.
- Desconsidere a integralidade das garrafas.

Exercício 1.9
A empresa de televisão "Boa vista" precisa decidir quantas TVs de 29"e 31"ela vai produzir. Uma analise do mercado descobriu que podem ser vendidas no máximo 40 TVs de 29"e 10 de 31"por mês. O trabalho máximo disponível por mês é 500h. A produção de um TV de 29"precisa 20h de trabalho, e um TV de 31"precisa 10h. Cada TV de 29"rende um lucro de R$ 120 e cada de 31"um lucro de R$ 80.
Qual a produção ótima média de cada TV por mês?

Exercício 1.10 (da Costa)
Um certo óleo é refinado a partir da mistura de outros óleos, vegetais ou não vegetais. Temos óleos vegetais V1 e V2 e óleos não vegetais NV1 NV2 NV3. Por restrições da fábrica, um máximo de 200 toneladas de óleos vegetais podem ser refinados por mês, e um máximo de 250 toneladas de óleos não vegetais. A acidez do óleo desejado deve estar entre 3 e 6 (dada uma unidade de medida) e a acidez depende linearmente das quantidades/acidez dos óleos brutos usados. O preço de venda de uma tonelada do óleo é R$ 150. Calcule a mistura que maximiza o lucro, dado que:

Óleo	V1	V2	NV1	NV2	NV3
Custo/ton	110	120	130	110	115
Acidez	8,8	6,1	2,0	4,2	5,0

Exercício 1.11 (Campêlo Neto)
Um estudante, na véspera de seus exames finais, dispõe de 100 horas de estudo para dedicar às disciplinas A, B e C. Cada um destes exames é formado por 100 questões, e o estudante espera acertar, alternativamente, uma questão em A, duas em B ou três em C, por cada hora de estudo. Suas notas nas provas anteriores foram 6, 7 e 10, respectivamente, e sua aprovação depende

de atingir uma média mínima de 5 pontos em cada disciplina. O aluno deseja distribuir seu tempo de forma a ser aprovado com a maior soma total de notas.

Exercício 1.12 (Dasgupta et al. (2009))
Moe está decidindo quanta cerveja Duff regular e quanta cerveja Duff Forte encomendar a cada semana. Duff regular custa a Moe $1 por caneco e ele a vende por $2 por caneco; Duff Forte custa $1.50 por caneco e ele vende por $3 por caneco. Entretanto, como parte de uma complicada fraude de marketing, a companhia Duff somente vende um caneco de Duff Forte para cada dois canecos ou mais de Duff regular que Moe compra. Além disso, devido a eventos passados sobre os quais é melhor nem comentar, Duff não venderá Moe mais do que 3000 canecos por semana. Moe sabe que ele pode vender tanta cerveja quanto tiver.
Formule um programa linear em duas variáveis para decidir quanto de Duff regular e quanto de Duff Forte comprar, para maximizar o lucro de Moe.

Exercício 1.13 (Dasgupta et al. (2009))
A companhia de produtos caninos oferece duas comidas para cachorro: Frisky Pup e Husky Hound, que são feitas de uma mistura de cereais e carne. Um pacote de Frisky Pup requer 1 quilo de cereal e 1.5 quilo de carne, e é vendido por $7. Um pacote de Husky Hound usa 2 quilos de cereal e 1 quilo de carne, e é vendido por $6. O cereal bruto custa $1 por quilo e a carne bruta, $2 por quilo. Há também o custo de $1.40 para empacotar o Frisky Pup e $0.60 para o Husky Hound. Um total de 240000 quilos de cereal e 180000 quilos de carne estão disponíveis a cada mês. O único gargalo de produção está no fato de a fábrica poder empacotar apenas 110000 pacotes de Frisky Pup por mês. Desnecessário dizer, a gerência gostaria de maximizar o lucro. Formule o problema como um programa linear em duas variáveis.

Exercício 1.14 (Vanderbei (2014))
Formule como problema de otimização linear e resolve graficamente.
Uma empresa de aço produz placas ou canos de ferro. As taxas de produção são 200t/h para placas e 140t/h para canos. O lucro desses produtos e 25$/t para placas e 30$/t para canos. Considerando a demanda atual, os limites de produção são 6000t de placas e 4000t de canos. Na semana atual são 40h de tempo de produção disponível. Quantas toneladas de placas e canos devem ser produzidas para maximizar o lucro?

Exercício 1.15 (Vanderbei (2014))
Formule como problema de otimização linear.
Uma pequena empresa aérea oferece um vôo de Pelotas, com escala em Porto Alegre para Torres. Logo tem três tipos de clientes que voam Pelotas–Porto Alegre, Pelotas–Torres e Porto Alegre–Torres. A linha também oferece três tipos de bilhetes:

- Tipo A: bilhete regular.
- Tipo B: sem cancelamento.
- Tipo C: sem cancelamento, pagamento três semanas antes de viajar.

Os preços (em R$) dos bilhetes são

	Pelotas–Porto Alegre	Porto Alegre–Torres	Pelotas–Torres
A	600	320	720
B	440	260	560
C	200	160	280

Baseado na experiência com esse vôo, o marketing tem a seguinte predição de passageiros:

	Pelotas–Porto Alegre	Porto Alegre–Torres	Pelotas–Torres
A	4	8	3
B	8	13	10
C	22	20	18

O objetivo da empresa e determinar o número ótimo de bilhetes para vender de cada tipo, respeitando um limite de 30 passageiros em cada vôo e o limite dos passageiros previstos em cada categoria, que maximiza o lucro.

Exercício 1.16
Resolva graficamente.

$$\begin{aligned}
\textbf{maximiza} \quad & 4x_1 + x_2 \\
\textbf{sujeito a} \quad & -x_1 + x_2 \leq 2, \\
& x_1 + 8x_2 \leq 36, \\
& x_2 \leq 4, \\
& x_1 \leq 4.25, \\
& x_1, x_2 \geq 0.
\end{aligned}$$

(a) Qual a solução ótima?

(b) Qual o valor da solução ótima?

Exercício 1.17
Escreve em forma normal.

$$\begin{aligned}
\textbf{minimiza} \quad & z = -5x_1 - 5x_2 - 5x_3 \\
\textbf{sujeito a} \quad & -6x_1 - 2x_2 - 9x_3 \leq 0, \\
& -9x_1 - 3x_2 + 3x_3 = 3, \\
& x_1, x_2, x_3 \geq 0.
\end{aligned}$$

$$\begin{aligned}
\textbf{maximiza} \quad & z = -6x_1 - 2x_2 - 6x_3 + 4x_4 + 4x_5 \\
\textbf{sujeito a} \quad & -3x_1 - 8x_2 - 6x_3 - 7x_4 - 5x_5 = 3, \\
& 5x_1 - 7x_2 + 7x_3 + 7x_4 - 6x_5 \leq 6, \\
& 1x_1 - 9x_2 + 5x_3 + 7x_4 - 10x_5 = -6, \\
& x_1, x_2, x_3, x_4, x_5 \geq 0.
\end{aligned}$$

$$\begin{aligned}
\textbf{maximiza} \quad & z = 7x_1 + 4x_2 + 8x_3 + 7x_4 - 9x_5 \\
\textbf{sujeito a} \quad & -4x_1 - 1x_2 - 7x_3 - 8x_4 + 6x_5 = -2, \\
& x_1 + 4x_2 + 2x_3 + 2x_4 - 7x_5 \geq -7, \\
& -8x_1 + 2x_2 + 8x_3 - 6x_4 - 7x_5 = -7, \\
& x_1, x_2, x_3, x_4, x_5 \geq 0.
\end{aligned}$$

$$
\begin{aligned}
\textbf{minimiza} \quad & z = -6x_1 + 5x_2 + 8x_3 + 7x_4 - 8x_5 \\
\textbf{sujeito a} \quad & -5x_1 - 2x_2 + x_3 - 9x_4 - 7x_5 = 9, \\
& 7x_1 + 7x_2 + 5x_3 - 3x_4 + x_5 = -8, \\
& -5x_1 - 3x_2 - 5x_3 + 9x_4 + 8x_5 \leq 0, \\
& x_1, x_2, x_3, x_4, x_5 \geq 0.
\end{aligned}
$$

2. O método Simplex

Graficamente, é difícil resolver sistemas com mais que três variáveis. Portanto é necessário achar métodos que permitam resolver sistemas grandes. Um dos mais importantes é o *método Simplex*. Nós vamos estudar esse método primeiramente através da aplicação a um exemplo.

2.1. Um exemplo

Começamos com o seguinte sistema em forma padrão:

Exemplo: Simplex

$$\begin{aligned} \textbf{maximiza} \quad & z = 6x_1 + 8x_2 + 5x_3 + 9x_4 \\ \textbf{sujeito a} \quad & 2x_1 + x_2 + x_3 + 3x_4 \leq 5, \\ & x_1 + 3x_2 + x_3 + 2x_4 \leq 3, \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned}$$

Introduzimos variáveis de folga e reescrevemos as equações:

Exemplo: Com variáveis de folga

$$\begin{aligned} \textbf{maximiza} \quad & z = 6x_1 + 8x_2 + 5x_3 + 9x_4 && (2.1) \\ \textbf{sujeito a} \quad & w_1 = 5 - 2x_1 - x_2 - x_3 - 3x_4, && (2.2) \\ & w_2 = 3 - x_1 - 3x_2 - x_3 - 2x_4, && (2.3) \\ & x_1, x_2, x_3, x_4, w_1, w_2 \geq 0. \end{aligned}$$

Observação 2.1
Nesse exemplo é fácil obter uma solução viável, escolhendo $x_1 = x_2 = x_3 = x_4 = 0$. Podemos verificar que $w_1 = 5$ e $w_2 = 3$ e todas as restrições são respeitadas. O valor da função objetivo seria 0. Uma outra solução viável é $x_1 = 1$, $x_2 = x_3 = x_4 = 0$, $w_1 = 3$, $w_2 = 2$ com valor $z = 6$. ◊

Com seis variáveis e duas equações lineares independentes o espaço de soluções do sistema de equações lineares dado pelas restrições tem $6-2=4$ graus de liberdade. Uma solução viável com esse número de *variáveis nulas* (igual a 0) se chama uma *solução básica viável*. Logo nossa primeira solução acima é uma solução básica viável.
A idéia do método Simplex é percorrer soluções básicas viáveis, aumentando em cada passo o valor z da função objetivo.
Logo nosso próximo objetivo é aumentar o valor da função objetivo z. Para esse fim, podemos aumentar o valor das variáveis x_1, x_2, x_3 ou x_4, pois o coeficiente delas é positivo. Escolhemos x_4, porque essa variável tem o maior coeficiente. Não podemos aumentar x_4 arbitrariamente: Para respeitar as restrições $w_1, w_2 \geq 0$ temos os limites

Limites

$$w_1 = 5 - 3x_4 \geq 0 \Longleftrightarrow x_4 \leq 5/3$$
$$w_2 = 3 - 2x_4 \geq 0 \Longleftrightarrow x_4 \leq 3/2$$

ou seja $x_4 \leq 3/2$. Aumentando x_4 o máximo possível, obtemos $x_4 = 3/2$ e $w_2 = 0$. Os valores das demais variáveis não mudam. Essa solução respeita novamente todas as restrições, e portanto é *viável*. Ainda, como trocamos uma variável nula (x_4) com uma outra não-nula (w_2) temos uma nova solução básica viável

Solução básica viável

$$x_1 = x_2 = x_3 = 0; x_4 = 3/2; w_1 = 1/2; w_2 = 0$$

com valor da função objetivo $z = 13.5$.
O que facilitou esse primeiro passo foi a forma especial do sistema de equações. Escolhemos quatro variáveis independentes (x_1, x_2, x_3 e x_4) e duas variáveis dependentes (w_1 e w_2). Essas variáveis são chamadas *não-básicas* e *básicas*, respectivamente. Na nossa solução básica viável todas variáveis não-básicas são nulas. Logo, pode-se aumentar uma variável não-básica cujo coeficiente na função objetivo seja positivo (para aumentar o valor da função objetivo). Inicialmente tem-se as seguintes variáveis básicas e não-básicas

$$\mathcal{B} = \{w_1, w_2\}; \qquad \mathcal{N} = \{x_1, x_2, x_3, x_4\}.$$

Depois de aumentar x_4 (e consequentemente zerar w_2) podemos escolher

$$\mathcal{B} = \{w_1, x_4\}; \qquad \mathcal{N} = \{x_1, x_2, x_3, w_2\}.$$

A variável x_4 se chama *variável entrante*, porque ela entra no conjunto de variáveis básicas B. Analogamente w_2 se chama *variável sainte*.
Para continuar, podemos reescrever o sistema atual com essas novas variáveis básicas e não-básicas. A segunda restrição 2.3 é fácil de reescrever

$$\begin{aligned} w_2 = 3 - x_1 - 3x_2 - x_3 - 2x_4 &\iff 2x_4 = 3 - x_1 - 3x_2 - x_3 - w_2 \\ &\iff x_4 = 3/2 - 1/2x_1 - 3/2x_2 - 1/2x_3 - 1/2w_2 \end{aligned}$$

Além disso, temos que reescrever a primeira restrição 2.2, porque a variável básica w_1 depende de x_4 que agora é básica também. Nosso objetivo é escrever todas variáveis básicas em termos de variáveis não-básicas. Para esse fim, podemos usar combinações lineares da linhas, que eliminam as variáveis não-básicas. Em nosso exemplo, a combinação (2.2)$-3/2$(2.3) elimina x_4 e resulta em

$$w_1 - 3/2w_2 = 1/2 - 1/2x_1 + 7/2x_2 + 1/2x_3$$

e colocando a variável não-básica w_2 no lado direito obtemos

$$w_1 = 1/2 - 1/2x_1 + 7/2x_2 + 1/2x_3 + 3/2w_2.$$

Temos que aplicar uma operação semelhante à função objetivo que ainda depende da variável básica x_4. Escolhemos (2.1)$-9/2$(2.3) para obter

$$z = 27/2 + 3/2x_1 - 11/2x_2 + 1/2x_3 - 9/2w_2.$$

Novo sistema

$$\begin{aligned} \textbf{maximiza} \quad & z = 27/2 + 3/2x_1 - 11/2x_2 + 1/2x_3 - 9/2w_2 \\ \textbf{sujeito a} \quad & w_1 = 1/2 - 1/2x_1 + 7/2x_2 + 1/2x_3 + 3/2w_2, \\ & x_4 = 3/2 - 1/2x_1 - 3/2x_2 - 1/2x_3 - 1/2w_2, \\ & x_1, x_2, x_3, x_4, w_1, w_2 \geq 0. \end{aligned}$$

que obtemos após uma operação de trocar as variáveis x_4 e w_2. Essa operação se chama um *pivô*. Observe que no novo sistema é fácil recuperar toda

informação atual: zerando as variáveis não-básicas obtemos diretamente a solução $x_1 = x_2 = x_3 = w_2 = 0$, $w_1 = 1/2$ e $x_4 = 3/2$ com função objetivo $z = 27/2$.
Antes de continuar "pivotando" introduzimos uma forma mais simples de escrever o sistema

Dicionário

$$\begin{array}{llllll} z & = 27/2 & +3/2x_1 & -11/2x_2 & +1/2x_3 & -9/2w_2 \\ \hline w_1 & = 1/2 & -1/2x_1 & +7/2x_2 & +1/2x_3 & +3/2w_2 \\ x_4 & = 3/2 & -1/2x_1 & -3/2x_2 & -1/2x_3 & -1/2w_2 \end{array}$$

que se chama *dicionário* (inglês: dictionary).

Excurso 2.1
Alguns autores usam um *tableau* em vez de um dicionário. Para n variáveis e m restrições, um tableau consiste em $n+1$ colunas e $m+1$ linhas. Igual a um dicionário, a primeira linha corresponde com a função objetivo, e as restantes linhas com as restrições. Diferente do dicionário a primeira coluna contém as constantes, e as restantes colunas correspondem com as variáveis, incluindo as básicas. Nosso exemplo acima em forma de tableau é

				base		
	x_1	x_2	x_3	x_4	w_1	w_2
27/2	3/2	−11/2	1/2	0	0	9/2
1/2	1/2	−7/2	−1/2	0	1	−3/2
3/2	1/2	3/2	1/2	1	0	1/2

◊

No próximo passo podemos aumentar somente x_1 ou x_3 porque somente elas têm coeficientes positivos. Aumentando x_1 temos que respeitar $x_1 \leq 1$ (da primeira restrição) e $x_1 \leq 3$ (da segunda). Logo a primeira restrição é mais forte, x_1 é a variável entrante, w_1 a variável sainte, e depois do pivô obtemos

Segundo passo

$$\begin{array}{llllll} z & = 15 & -3w_1 & +5x_2 & +2x_3 & \\ \hline x_1 & = 1 & -2w_1 & +7x_2 & +x_3 & +3w_2 \\ x_4 & = 1 & +w_1 & -5x_2 & -x_3 & -2w_2 \end{array}$$

No próximo pivô x_2 entra. A primeira restrição não fornece limite para x_2, porque o coeficiente de x_2 é positivo! Mas a segunda $x_2 \leq 1/5$ e x_4 sai da base. O resultado do pivô é

Terceiro passo

z	$= 16$	$-2w_1$	$-x_4$	$+x_3$	$-2w_2$
x_1	$= 12/5$	$-3/5w_1$	$-7/5x_4$	$-2/5x_3$	$+1/5w_2$
x_2	$= 1/5$	$+1/5w_1$	$-1/5x_4$	$-1/5x_3$	$-2/5w_2$

O próximo pivô: x_3 entra, x_2 sai:

Quarto passo

z	$= 17$	$-w_1$	$-2x_4$	$-5x_2$	$-4w_2$
x_1	$= 2$	$-w_1$	$-x_4$	$+2x_2$	$+w_2$
x_3	$= 1$	$+w_1$	$-x_4$	$-5x_2$	$-2w_2$

Agora, todos coeficientes da função objetivo são negativos. Isso significa, que não podemos mais aumentar nenhuma variável não-básica. Como esse sistema é equivalente ao sistema original, qualquer solução tem que ter um valor menor ou igual a 17, pois todas as variáveis são positivas. Logo chegamos no resultado final: a solução

$$w_1 = x_4 = x_2 = w_2 = 0; x_1 = 2; x_3 = 1$$

com valor objetivo 17, é ótima!
Concluímos esse exemplo com mais uma observação. O número de soluções básicas viáveis é limitado. Em nosso exemplo, se escolhemos um subconjunto de quatro variáveis nulas, as duas equações determinam as variáveis restantes. Logo temos no máximo $\binom{6}{4} = 15$ soluções básicas viáveis. Em geral, com m equações e n variáveis, uma solução básica viável possui $n - m$ variáveis nulas e o número delas é limitado por $\binom{n}{n-m}$. Portanto, se aumentamos em cada pivô o valor da função objetivo, o método termina em no máximo $\binom{n}{n-m}$ passos.

Exemplo 2.1 (Solução do problema do Ildo)
Exemplo da solução do problema do Ildo na página 7.

$z =$	$0/1$	$+1/5c$	$+1/2s$
$w_1 =$	150	$-c$	$-3/2s$
$w_2 =$	6000	$-50c$	$-50s$
$w_3 =$	80	$-c$	
$w_4 =$	60		$-s$

Pivô s–w_4

$z =$	30	$+1/5c$	$-1/2w_4$
$w_1 =$	60	$-c$	$+3/2w_4$
$w_2 =$	3000	$-50c$	$+50w_4$
$w_3 =$	80	$-c$	
$s =$	60		$-w_4$

Pivô c–w_1

$z =$	42	$-1/5w_1$	$-1/5w_4$
$c =$	60	$-w_1$	$+3/2w_4$
$w_2 =$		$+50w_1$	$-25w_4$
$w_3 =$	20	$+w_1$	$-3/2w_4$
$s =$	60		$-w_4$

O resultado é um lucro total de R$ 42, com os seguintes valores de variáveis: $c = 60$, $s = 60$, $w_1 = 0$, $w_2 = 0$, $w_3 = 20$ e $w_4 = 0$. A interpretação das variáveis de folga é como segue.

- w_1: Número de ovos sobrando: 0.
- w_2: Quantidade de açúcar sobrando: 0 g.
- w_3: Croissants não produzidos (abaixo da demanda): 20.
- w_4: Strudels não produzidos: 0.

$\Diamond$

2.2. O método resumido

Considerando n variáveis e m restrições:

Sistema inicial

$$\begin{aligned} \textbf{maximiza} \quad & z = \sum_{j\in[n]} c_j x_j \\ \textbf{sujeito a} \quad & \sum_{j\in[n]} a_{ij} x_j \le b_i, && i \in [m], \\ & x_j \ge 0, && j \in [n]. \end{aligned}$$

Preparação
Introduzimos variáveis de folga

$$\sum_{j\in[n]} a_{ij} x_j + x_{n+i} = b_i, \qquad i \in [m],$$

e escrevemos as variáveis de folga como dependentes das variáveis restantes

$$x_{n+i} = b_i - \sum_{j\in[n]} a_{ij} x_j, \qquad i \in [m].$$

Solução básica viável inicial
Se todos $b_i \ge 0$ (o caso contrário vamos tratar na próxima seção), temos uma solução básica inicial

$$\begin{aligned} x_{n+i} &= b_i, \qquad i \in [m], \\ x_j &= 0, \qquad j \in [n]. \end{aligned}$$

Índices das variáveis
Depois do primeiro passo, os conjuntos de variáveis básicas e não-básicas mudam. Seja $\mathcal{B}$ o conjunto dos índices das variáveis básicas (não-nulas) e $\mathcal{N}$ o conjunto das variáveis nulas. No começo temos

$$\mathcal{B} = \{n+1, n+2, \ldots, n+m\}; \qquad \mathcal{N} = \{1, 2, \ldots, n\}$$

A forma geral do sistema muda para

$$z = \bar{z} + \sum_{j \in \mathcal{N}} \bar{c}_j x_j,$$
$$x_i = \bar{b}_i - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j, \qquad i \in \mathcal{B}.$$

As barras em cima dos coeficientes enfatizam que eles mudam ao longo da aplicação do método. Os coeficientes $\bar{c}_j$ são chamados *custos reduzidos* (ingl. reduced costs).

Escolher variável entrante (ingl. pricing)
Em cada passo do método Simplex, escolhemos uma variável não-básica x_k, com $k \in \mathcal{N}$ para aumentar o valor objetivo z. Isso somente é possível para os índices j tal que $\bar{c}_j > 0$, i.e.

$$\{j \in \mathcal{N} \mid \bar{c}_j > 0\}. \tag{2.4}$$

Escolhemos um k desse conjunto, e x_k é a variável entrante. Uma heurística simples é a *regra do maior coeficiente*, que escolhe

$$k = \operatorname{argmax}\{\bar{c}_j \mid \bar{c}_j > 0, j \in \mathcal{N}\}$$

Aumentar a variável entrante
Seja x_k a variável entrante. Se aumentarmos a variável x_k para um valor positivo, as variáveis básicas assumem novos valores

$$x_i = \bar{b}_i - \bar{a}_{ik} x_k \qquad i \in \mathcal{B}.$$

Temos que respeitar $x_i \geq 0$ para $1 \leq i \leq n$. Cada equação com $\bar{a}_{ik} > 0$ fornece uma cota superior

$$x_k \leq \bar{b}_i / \bar{a}_{ik}$$

para o aumento de x_k. Logo podemos aumentar x_k no máximo por um valor

$$\alpha := \min_{i \in \mathcal{B} \mid \bar{a}_{ik} > 0} \bar{b}_i / \bar{a}_{ik} = \left(\max_{i \in \mathcal{B} \mid \bar{a}_{ik} > 0} \bar{a}_{ik} / \bar{b}_i \right)^{-1} = \left(\max_{i \in \mathcal{B}} \bar{a}_{ik} / \bar{b}_i \right)^{-1} > 0. \tag{2.5}$$

É possível que múltiplas variáveis definem o mesmo limite: podemos escolher a variável sainte entre os índices

$$\{i \in \mathcal{B} \mid \bar{b}_i / \bar{a}_{ik} = \alpha\}. \tag{2.6}$$

2.3. Sistemas ilimitados

Como pivotar?

- Considere o sistema

$$\begin{array}{llll} z & = 24 & -x_1 & +2x_2 \\ \hline x_3 & = 2 & -x_1 & +x_2 \\ x_4 & = 5 & +x_1 & +4x_2 \end{array}$$

- Qual a próxima solução básica viável?
- A duas equações não restringem o aumento de x_2: existem soluções com valor ilimitado.

2.4. Encontrar uma solução inicial: o método de duas fases

Solução básica inicial

- Nosso problema inicial é

$$\begin{aligned} \textbf{maximiza} \quad & z = \sum_{j \in [n]} c_j x_j \\ \textbf{sujeito a} \quad & \sum_{j \in [n]} a_{ij} x_j \leq b_i, && i \in [m], \\ & x_i \geq 0, && i \in [n], \end{aligned}$$

- com dicionário inicial

$$\begin{aligned} z &= \bar{z} + \sum_{j \in \mathcal{N}} \bar{c}_j x_j \\ x_i &= \bar{b}_i - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j, \qquad i \in \mathcal{B}. \end{aligned}$$

Solução básica inicial

- A solução básica inicial desse dicionário é

$$x = (0 \cdots 0\, b_1 \cdots b_m)^t$$

- O que acontece se existe um $b_i < 0$?
- A solução básica não é mais viável! Sabe-se disso porque pelo menos uma variável básica terá valor negativo.

Sistema auxiliar

- Um método para resolver o problema: resolver outro programa linear
  - cuja solução fornece uma solução básica viável do programa linear original e
  - que tem uma solução básica viável simples, tal que podemos aplicar o método Simplex.

$$\begin{aligned} \textbf{maximiza} \quad & z = -x_0 \\ \textbf{sujeito a} \quad & \sum_{j \in [n]} a_{ij} x_j - x_0 \leq b_i, && 0 \leq i \leq m, \\ & x_i \geq 0, && i \in [n]. \end{aligned}$$

Resolver o sistema auxiliar

- É fácil encontrar uma solução viável do sistema auxiliar:
  - Escolhe $x_i = 0$, para todos $i \in [n]$.
  - Escolhe x_0 suficientemente grande: $x_0 \geq \max_{i \in [m]} -b_i$.
- Isso corresponde com um primeiro pivô com variável entrante x_0 após introduzir as variáveis de folga ("pseudo-pivô").
  - Podemos começar com a solução não-viável $x_0 = x_1 = \ldots = x_n = 0$.
  - Depois aumentamos x_0 tal que a variável de folga mais negativa vire positiva.
  - x_0 e variável sainte x_k tal que $k = \operatorname{argmax}_{i \in [m]} -b_i$.

Exemplo: Problema original

$$\begin{aligned} \textbf{maximiza} \quad & z = -2x_1 - x_2 \\ \textbf{sujeito a} \quad & -x_1 + x_2 \leq -1, \\ & -x_1 - 2x_2 \leq -2, \\ & x_2 \leq 1, \\ & x_1, x_2 \geq 0. \end{aligned}$$

Exemplo: Problema auxiliar

$$\begin{aligned} \textbf{maximiza} \quad & z = -x_0 \\ \textbf{sujeito a} \quad & -x_1 + x_2 - x_0 \leq -1, \\ & -x_1 - 2x_2 - x_0 \leq -2, \\ & x_2 - x_0 \leq 1, \\ & x_0, x_1, x_2 \geq 0. \end{aligned}$$

Exemplo: Dicionário inicial do problema auxiliar

z	$=$			$-x_0$
w_1	$=-1$	$+x_1$	$-x_2$	$+x_0$
w_2	$=-2$	$+x_1$	$+2x_2$	$+x_0$
w_3	$=1$		$-x_2$	$+x_0$

- Observe que a solução básica não é viável.
- Para achar uma solução básica viável: fazemos um primeiro pivô com variável entrante x_0 e variável sainte w_2.

Exemplo: Dicionário inicial viável do sistema auxiliar

z	$=-2$	$+x_1$	$+2x_2$	$-w_2$
w_1	$=1$		$-3x_2$	$+w_2$
x_0	$=2$	$-x_1$	$-2x_2$	$+w_2$
w_3	$=3$	$-x_1$	$-3x_2$	$+w_2$

Primeiro pivô

z	$=-4/3$	$+x_1$	$-2/3w_1$	$-1/3w_2$
x_2	$=1/3$		$-1/3w_1$	$+1/3w_2$
x_0	$=4/3$	$-x_1$	$+2/3w_1$	$+1/3w_2$
w_3	$=2$	$-x_1$	$+w_1$	

Segundo pivô

z	$= 0$	$-x_0$		
x_2	$= 1/3$		$-1/3w_1$	$+1/3w_2$
x_1	$= 4/3$	$-x_0$	$+2/3w_1$	$+1/3w_2$
w_3	$= 2/3$	$+x_0$	$+1/3w_1$	$-1/3w_2$

Solução ótima!

Solução do sistema auxiliar

- O que podemos concluir da solução do sistema auxiliar?
- Obviamente, se o sistema original possui solução, o sistema auxiliar também possui uma solução com $x_0 = 0$.
- Logo, após aplicar o método Simplex ao sistema auxiliar, temos os casos
  - $x_0 > 0$: O sistema original não tem solução.
  - $x_0 = 0$: O sistema original tem solução. Podemos descartar x_0 e continuar resolvendo o sistema original com a solução básica viável obtida.
- A solução do sistema auxiliar se chama *fase I*, a solução do sistema original *fase II*.

Sistema original

Reescreve-se a função objetivo original substituindo as variáveis básicas do sistema original pelas equações correspondentes do sistema auxiliar, de forma que a função objetivo z não contenha variáveis básicas. No exemplo, a função objetivo é rescrita como:

$$z = -2x_1 - x_2 = -3 - w_1 - w_2.$$

z	$= -3$	$-w_1$	$-w_2$
x_2	$= 1/3$	$-1/3w_1$	$+1/3w_2$
x_1	$= 4/3$	$+2/3w_1$	$+1/3w_2$
w_3	$= 2/3$	$+1/3w_1$	$-1/3w_2$

Nesse exemplo, o dicionário original já é ótimo!

Exemplo 2.2 (Sistema original inviável)
O sistema

$$\begin{aligned} \textbf{maximiza} \quad & x_1 + x_2 \\ \textbf{sujeito a} \quad & x_1 + x_2 \geq 2, \\ & x_1 + x_2 \leq 1, \\ & x_1, x_2 \geq 0. \end{aligned}$$

obviamente não possui uma solução viável. O dicionário inicial do sistema auxiliar (após normalização e introdução das variáveis de folga) é

$$\begin{array}{rrrrr} z = & 0 & & & -x_0 \\ \hline x_3 = & -2 & +x_1 & +x_2 & +x_0 \\ x_4 = & 1 & -x_1 & -x_2 & +x_0 \end{array}$$

e o pseudo-pivô x_0–x_3 produz

$$\begin{array}{rrrrr} z = & -2 & +x_1 & +x_2 & -x_3 \\ \hline x_0 = & 2 & -x_1 & -x_2 & +x_3 \\ x_4 = & 3 & -2x_1 & -2x_2 & +x_3 \end{array}$$

e o pivô x_1–x_4 produz o sistema ótimo

$$\begin{array}{rrrrr} z = & -1/2 & -1/2x_4 & & -1/2x_3 \\ \hline x_0 = & 1/2 & +1/2x_4 & & +1/2x_3 \\ x_1 = & 3/2 & -1/2x_4 & -x_2 & +1/2x_3 \end{array} .$$

O valor ótimo do sistema auxiliar é $-z = x_0 = 1/2$, confirmando que o sistema original não possui solução viável. ◊

2.4.1. Resumo do método de duas fases

Fase I necessária? Caso $b_i \geq 0$ para todo $i \in [m]$: continua com a fase II.

Dicionário inicial Cria o dicionário inicial do sistema auxiliar

$$z = \min\{x_0 \mid Ax \leq b + x_o e\}.$$

Pseudo-pivô Pivota x_0–x_k, sendo $k = \operatorname{argmin}_{i \in [m]} b_k$ o índice do lado direito mais negativo.

Solução fase I Aplica o método no dicionário obtido no passo anterior.

Fase II necessária? Caso a solução ótima da fase I possui valor $x_0 > 0$: o sistema original não possui solução. Para.

Prepara fase II Caso x_0 é uma variável básica: pivota x_0–x_k sendo x_k alguma variável nula tal que $\overline{a}_{0k} \neq 0$. Remove a coluna x_0. Remove a função objetivo do sistema auxiliar e introduz a função objetivo do sistema original (escrita em função das variáveis nulas).

Fase II Aplica o método Simplex no dicionário inicial da fase II.

2.5. Sistemas degenerados

Sistemas, soluções e pivôs degenerados

- Um dicionário é *degenerado* se existe um $i \in \mathcal{B}$ tal que $\bar{b}_i = 0$.
- Qual o problema?
- Pode acontecer um pivô que não aumenta a variável entrante, e portanto não aumenta o valor da função objetivo.
- Tais pivôs são *degenerados*.

Exemplo 1

- Nem sempre é um problema.

$$\begin{array}{llll} z & = 5 & +x_3 & -x_4 \\ \hline x_2 & = 5 & -2x_3 & -3x_4 \\ x_1 & = 7 & & -4x_4 \\ w_3 & = 0 & & +x_4 \end{array}$$

- x_2 é a variável sainte e o valor da função objetivo aumenta.

Exemplo 2

$$\begin{array}{llllll} z & = 3 & -1/2x_1 & +2x_2 & -3/2w_1 \\ \hline x_3 & = 1 & -1/2x_1 & & -1/2w_1 \\ w_2 & = 0 & +\,x_1 & -x_2 & +w_1 \end{array}$$

- Se a variável sainte é determinada pela equação com $\bar{b}_i = 0$, temos um *pivô degenerado*.

- Nesse caso, a variável entrante não aumenta: temos a mesma solução depois do pivô.

Exemplo 2: Primeiro pivô

- Pivô: x_2–w_2

$$\begin{array}{llllll} z & = 3 & +3/2x_1 & -2w_2 & +1/2w_1 \\ \hline x_3 & = 1 & -1/2x_1 & & -1/2w_1 \\ x_2 & = 0 & +x_1 & -w_2 & +w_1 \end{array}$$

- O valor da função objetivo não aumentou!

Exemplo 2: Segundo pivô

- Pivô: x_1–x_3

$$\begin{array}{llllll} z & = 6 & -3x_3 & -2w_2 & -w_1 \\ \hline x_1 & = 2 & -2x_3 & & -w_1 \\ x_2 & = 2 & -2x_3 & -w_2 & \end{array}$$

- A segunda iteração aumentou o valor da função objetivo!

Ciclos

- O pior caso seria, se entramos em ciclos.
- É possível? Depende da regra de seleção de variáveis entrantes e saintes.
- Nossas regras
  - Escolhe a variável entrante com o maior coeficiente.
  - Escolhe a variável sainte mais restrita.
  - Em caso de empate, escolhe a variável com o menor índice.
- Ciclos são possíveis: O seguinte sistema possui um ciclo de seis pivôs: x_1–w_1, x_2–w_2, x_3–x_1, x_4–x_2, w_1–x_3, w_2–x_4.

$$\begin{array}{llllll} z & = & 10x_1 & -57x_2 & -9x_3 & -24x_4 \\ \hline w_1 & = 0 & -1/2x_1 & +11/2x_2 & +5/2x_3 & -9x_4 \\ w_2 & = 0 & -1/2x_1 & +3/2x_2 & +1/2x_3 & -x_4 \\ w_3 & = 1 & -x_1 & & & \end{array}$$

Soluções do problema

- Como resolver o problema?
- Três soluções
  - Ignora o problema (ou perturba numericamente).
  - Método lexicográfico (perturba simbolicamente).
  - Regra de Bland.

Método lexicográfico

- Idéia: O fato que existe um $\bar{b}_i = 0$ é por acaso.
- Se introduzimos uma pequena perturbação $\epsilon \ll 1$
  - o problema desaparece
  - a solução será (praticamente) a mesma.

Método lexicográfico

- Ainda é possível que duas perturbações numéricas se cancelem.
- Para evitar isso: Trabalha-se simbolicamente.
- Introduzimos perturbações simbólicas

$$\text{const.} \gg \epsilon_1 \gg \epsilon_2 \gg \cdots \gg \epsilon_m > 0$$

  em cada equação.

- Característica: Todo ϵ_i é numa escala diferente dos outros tal que eles não se cancelam.

Exemplo

Exemplo 2.3
Sistema original degenerado e sistema perturbado

z	$= 4$	$+2x_1$	$-x_2$
w_1	$= 1/2$		$-x_2$
w_2	$= 0$	$-2x_1$	$+4x_2$
w_3	$= 0$	$+x_1$	$-3x_2$

z	$= 4$					$+2x_1$	$-x_2$
w_1	$= 1/2$	$+\epsilon_1$					$-x_2$
w_2	$= 0$		$+\epsilon_2$			$-2x_1$	$+4x_2$
w_3	$= 0$			$+\epsilon_3$		$+x_1$	$-3x_2$

$\Diamond$

Comparar perturbações

- Com variável entrante x_k, a linha de menor limite $\bar{b}_i/\bar{a}_{ik}$ com $\bar{a}_{ik} > 0$ define a variável sainte.
- Os coeficientes agora contem constantes e perturbações.
- Podemos representá-los como vetores $\bar{b}_i^t\epsilon$ com

$$\bar{b}_i^t = (\bar{v}_i\, e_{i1} \cdots, e_{im})$$

  e com $\epsilon = (1\,\epsilon_1 \ldots, \epsilon_m)^t$).
- Com isso temos limites $l_i = \bar{b}_i/\bar{a}_{ik}$ em cada linha i.
- A comparação de limites respeita a ordem lexicográfica das perturbações: $l_i < l_j$ sse o primeiro coeficiente não-nulo em $l_i - l_j$ é negativo.

Características

- Depois de chegar no valor ótimo, podemos retirar as perturbações ϵ_i.

  Teorema 2.1
  O método Simplex sempre termina escolhendo as variáveis saintes usando a regra lexicográfica.

Prova. É suficiente mostrar que o sistema nunca será degenerado. Neste caso o valor da função objetivo sempre cresce, e o método Simplex não cicla. A matriz de perturbações

$$\begin{pmatrix} \epsilon_1 & & & \\ & \epsilon_2 & & \\ & & \cdots & \\ & & & \epsilon_m \end{pmatrix}$$

inicialmente tem posto m. As operações do método Simplex são operações lineares que não mudam o posto do matriz. Logo, em cada passo do método Simplex temos uma matriz de perturbações

$$\begin{pmatrix} e_{11}\epsilon_1 & e_{12}\epsilon_2 & \cdots & e_{1m}\epsilon_m \\ e_{21}\epsilon_1 & e_{22}\epsilon_2 & \cdots & e_{2m}\epsilon_m \\ \cdots & & \cdots & \\ e_{m1}\epsilon_1 & e_{m2}\epsilon_2 & \cdots & e_{mm}\epsilon_m \end{pmatrix}$$

que ainda tem posto m. Portanto, em cada linha i existe pelo menos um $e_{ij} \neq 0$ e assim uma perturbação diferente de zero e o sistema não é degenerado. ■

Exemplo 2.4

Solução do exemplo 2.3.

Pivô x_1–w_3:

z	$= 4$		$+1\epsilon_2$		$-w_2$	$+3x_2$
w_1	$= 1/2$	$+\epsilon_1$				$-x_2$
x_1	$= 0$		$+1/2\epsilon_2$	$+2\epsilon_3$	$-1/2w_2$	$-2x_2$
w_3	$= 0$		$+1/2\epsilon_2$	$+\epsilon_3$	$-1/2w_2$	$+3x_2$

Pivô x_2–w_3:

z	$= 4$		$+5/2\epsilon_2$	$+3\epsilon_3$	$-5/2w_2$	$-3w_3$
w_1	$= 1/2$	$+\epsilon_1$	$-1/2\epsilon_2$	$-\epsilon_3$	$+1/2w_2$	$+w_3$
x_1	$= 0$		$+3/2\epsilon_2$	$+2\epsilon_3$	$-3/2w_2$	$-2w_3$
x_2	$= 0$		$+1/2\epsilon_2$	$+\epsilon_3$	$-1/2w_2$	$-w_3$

◊

Regra de Bland

- Outra solução do problema: A regra de Bland.
- Escolhe como variável entrante e sainte sempre a variável com o menor índice (caso tiver mais que um candidato).

  Teorema 2.2

  O método Simplex sempre termina se as variáveis entrantes e saintes são escolhidas através da regra de Bland.

Prova. Prova por contradição: Suponha que exista uma sequência de dicionários que entra num ciclo $D_0, D_1, \ldots, D_{k-1}$ usando a regra do Bland. Nesse ciclo algumas variáveis, chamadas *instáveis*, entram e saem novamente da base, outras permanecem sempre como básicas, ou como não-básicas. Seja x_t a variável instável com o maior índice. Sem perda de generalidade, seja x_t a variável sainte do primeiro dicionário D_0. Seja x_s a variável entrante no D_0. Observe que x_s também é instável e portanto $s < t$. Seja D^* o dicionário em que x_t entra na base. Temos a situação

$$\begin{array}{ccccccccc} x_s \text{ entra} & & & & x_t \text{ entra} & & & & \\ \downarrow & & & & \downarrow & & & & \\ D_0, & D_1, & D_2, & \cdots & D^*, & \cdots & D_{k-1} & & \\ \downarrow & & & & & & & & \\ x_t \text{ sai} & & & & & & & & \end{array}$$

com os sistemas correspondentes

$$\begin{array}{ll} D_0: & D^*: \\ z = z_0 + \sum_{j \in \mathcal{N}} c_j x_j & z = z^* + \sum_{j \in \mathcal{N}^*} c_j^* x_j \\ x_i = b_i - \sum_{j \in \mathcal{N}} a_{ij} x_j \quad i \in \mathcal{B} & x_i = b_i^* - \sum_{j \in \mathcal{N}^*} a_{ij}^* x_j \quad i \in \mathcal{B}^* \end{array}$$

Como temos um ciclo, todas variáveis instáveis tem valor 0 e o valor da função objetivo é constante. Logo $z_0 = z^*$ e para D^* temos

$$z = z^* + \sum_{j \in \mathcal{N}^*} c_j^* x_j = z_0 + \sum_{j \in \mathcal{N}^*} c_j^* x_j. \tag{2.7}$$

Se aumentamos em D_0 o valor do x_s para y, qual é o novo valor da função objetivo? Os valores das variáveis são

$$\begin{aligned} x_s &= y \\ x_j &= 0 \qquad j \in \mathcal{N} \setminus \{s\} \\ x_i &= b_i - a_{is} y \qquad i \in \mathcal{B} \end{aligned} \tag{2.8}$$

e temos no sistema D_1 o novo valor

$$z = z_0 + c_s y \tag{2.9}$$

Vamos substituir os valores das variáveis (2.8) com índices em $\mathcal{N}^* \cap \mathcal{B}$ na equação (2.7). Para facilitar a substituição, vamos definir $c_j^* := 0$ para $j \notin \mathcal{N}^*$, que permite substituir todas variáveis $x_j, j \in \mathcal{B}$ e assim obtemos

$$z = z_0 + \sum_{j \in [1,n+m]} c_j^* x_j = z_0 + c_s^* y + \sum_{j \in \mathcal{B}} c_j^* (b_j - a_{js} y). \tag{2.10}$$

Equações (2.9) e (2.10) representam o mesmo valor, portanto

$$\left(c_s - c_s^* + \sum_{j \in \mathcal{B}} c_j^* a_{js} \right) y = \sum_{j \in \mathcal{B}} c_j^* b_j.$$

Essa igualdade deve ser correta para qualquer aumento y, portanto os dois lados são 0, em particular

$$c_s - c_s^* + \sum_{j \in \mathcal{B}} c_j^* a_{js} = 0.$$

Como x_s entra em D_0 temos $c_s > 0$. Em D^* a variável x_t entra, então $c_s^* \leq 0$ senão pela regra de Bland $s < t$ entraria. Logo,

$$\sum_{j \in \mathcal{B}} c_j^* a_{js} = c_s^* - c_s \leq -c_s < 0$$

e deve existir um $r \in \mathcal{B}$ tal que $c_r^* a_{rs} < 0$. Isso tem uma série de consequências:

(i) $c_r^* \neq 0$.

(ii) $r \in \mathcal{N}^*$, porque somente as variáveis nulas satisfazem $c_j^* \neq 0$ em D^*.

(iii) x_r é instável, porque ela é básica em D_0 ($r \in \mathcal{B}$), mas não-básica em D^* ($r \in \mathcal{N}^*$).

(iv) $r \leq t$, porque t foi a variável instável com o maior índice.

(v) $r < t$, porque $c_t^* a_{ts} > 0$: x_t entra em D^*, logo $c_t^* > 0$, e x_t sai em D_0, logo $a_{ts} > 0$.

(vi) $c_r^* \leq 0$, senão r e não t entraria em D^* seguindo a regra de Bland.

(vii) $a_{rs} > 0$.

(viii) $b_r = 0$, porque x_r é instável, mas todos variáveis instáveis tem valor 0 no ciclo, e x_r é básica em D_0.

Os últimos dois itens mostram que x_r foi candidato ao sair em D_0 com índice $r < t$, uma contradição com a regra de Bland. ■

Teorema fundamental

Teorema 2.3 (Teorema fundamental da programação linear)
Para qualquer programa linear temos:

(i) Se não existe solução ótima, o problema é inviável ou ilimitado.

(ii) Se existe uma solução viável, existe uma solução básica viável.

(iii) Se existe uma solução ótima, existe uma solução ótima básica.

2.6. Complexidade do método Simplex

Usando a regra de Bland o método Simplex nunca repete uma base e o número de pivôs é limitado pelo número de bases. Com $n+m$ variáveis (de decisão e de folga) existem no máximo

$$\binom{n+m}{n} = \binom{n+m}{m}$$

bases possíveis. Para $n+m$ constante, essa expressão é maximizada para $n=m$. Os limites nesse caso são (exercício 2.3)

$$\frac{1}{2n}2^{2n} \leq \binom{2n}{n} \leq 2^{2n}.$$

Logo é possível que o método Simplex precisa um número exponencial de pivôs. A existência de sistemas com um número de pivôs exponencial depende da regra de pivoteamento. Por exemplo, para a regra de maior coeficiente, existem sistemas que precisam um número exponencial de pivôs (Klee-Minty). A pergunta se isso é o caso para qualquer regra de pivoteamento está em aberto. O melhor algoritmo para a programação linear precisa tempo $O((n^3/\log n)L$ (Anstreicher 1999), supondo que uma operação aritmética custa $O(1)$ e os dados são inteiros de L bits. Empiricamente o método Simplex precisa $O(m+n)$ pivôs (Vanderbei 2014), e cada pivô custa $O(mn)$ operações, logo o tempo empírico, novamente supondo que uma operação aritmética custa $O(1)$ do método Simplex é $O((m+n)mn)$.

Observação 2.2
Spielman e Teng (2004) mostram que o método Simplex possui *complexidade suavizada* polinomial, i.e., o máximo do valor esperado do tempo de execução sobre pequenos perturbações (Gaussianas) é polinomial no tamanho da instância e no inverso da perturbação.

Sem perturbações o problema de encontrar a solução que o método Simplex encontraria usando a regra de Dantzig é PSPACE-completo (Fearnley e Savani 2014). ◊

2.7. Exercícios

(Soluções a partir da página 218.)

Exercício 2.1 (Maculan e Fampa (2006))
Resolve com o método Simplex.

$$\begin{aligned} \textbf{maximiza} \quad & z = 3x_1 + 5x_2 \\ \textbf{sujeito a} \quad & x_1 \leq 4, \\ & x_2 \leq 6, \\ & 3x_1 + 2x_2 \leq 18, \\ & x_1, x_2 \geq 0. \end{aligned}$$

Exercício 2.2
Resolve o exercício 1.7 usando o método Simplex.

Exercício 2.3
Prova que

$$\frac{2^{2n}}{2n} \leq \binom{2n}{n} \leq 2^{2n}.$$

Exercício 2.4
Resolve o sistema degenerado

$$\begin{array}{llllll} z & = & 10x_1 & -57x_2 & -9x_3 & -24x_4 \\ \hline w_1 & = & -1/2x_1 & +11/2x_2 & +5/2x_3 & -9x_4 \\ w_2 & = & -1/2x_1 & +3/2x_2 & +1/2x_3 & -x_4 \\ w_3 & = 1 & -x_1 & & & \end{array}$$

usando o método lexicográfico e a regra de Bland.

Exercício 2.5
Dado o problema de otimização

$$\begin{aligned} \textbf{maximiza} \quad & x_1 + x_2 \\ \textbf{sujeito a} \quad & ax_1 + bx_2 \leq 1, \\ & x_1, x_2 \geq 0, \end{aligned}$$

determine condições suficientes e necessárias que a e b tem que satisfazer tal que

(a) existe pelo menos uma solução ótima,

(b) existe exatamente uma solução ótima,

(c) existe nenhuma solução ótima,

(d) o sistema é ilimitado.

ou demonstre que o caso não é possível.

Exercício 2.6
Sabe-se que o dicionário ótimo do problema

$$\begin{aligned} \textbf{maximiza} \quad & z = 3x_1 + x_2 \\ \textbf{sujeito a} \quad & -2x_1 + 3x_2 \leq 5, \\ & x_1 - x_2 \leq 1, \\ & x_1, x_2 \geq 0, \end{aligned}$$

é

$$\begin{array}{lll} z^* = 31 & -11w_2 & -4w_1 \\ \hline x_2 = 7 & -2w_2 & -w_1 \\ x_1 = 8 & -3w_2 & -w_1 \end{array}$$

(a) Se a função objetivo passar a $z = x_1 + 2x_2$, a solução continua ótima? No caso de resposta negativa, determine a nova solução ótima.

(b) Se a função objetivo passar a $z = x_1 - x_2$, a solução continua ótima? No caso de resposta negativa, determine a nova solução ótima.

(c) Se a função objetivo passar a $z = 2x_1 - 2x_2$, a solução continua ótima?No caso de resposta negativa, determine a nova solução ótima.

(d) Formular o dual e obter a solução dual ótima.

Exercício 2.7
Prove ou mostre um contra-exemplo.
O problema $\max\{c^t x \mid Ax \leq b\}$ possui uma solução viável sse $\min\{x_0 \mid Ax - ex_0 \leq b\}$ possui uma solução viável com $x_0 = 0$. Observação: e é um vetor com todos compentes igual 1 da mesma dimensão que b.

Exercício 2.8
Prove ou mostre um contra-exemplo.
Se x é a variável sainte em um pivô, x não pode ser variável entrante no pivô seguinte.

Exercício 2.9
Demonstramos na seção 2.5 que existem sistemas em que o método Simplex entra em ciclos. No exemplo o método Simplex ficou sempre na mesma solução, representada por bases diferentes. Agora supõe que temos soluções diferentes com o mesmo valor da função objetivo. É possível que o método Simplex entra num ciclo sempre visitando soluções diferentes?

Exercício 2.10
Supõe que temos um dicionário com uma base infactível, com um candidato para a variável entrante x_e (i.e. $c_e > 0$) tal que todos coeficientes na coluna correspondente são negativos (i.e. $a_{ie} < 0$ para todo $i \in \mathcal{B}$). Caso a base fosse viável podemos concluir que o sistema é ilimitado. Podemos concluir isso também com a base infactível?

3. Dualidade

3.1. Introdução

Visão global

- *Dualidade*: Cada programa linear (chamada de *primal*) possui um programa linear correspondente, chamado de *dual*.
- A dualidade tem várias aplicações como
  - Estimar a qualidade de soluções e a convergência do método Simplex.
  - Certificar a otimalidade de um programa linear.
  - Analisar a sensibilidade e re-otimizar sistemas.
  - Resolver programas lineares mais eficiente com o Método Simplex dual.
- O programa linear dual possui uma interpretação relevante.

Introdução

- Considere o programa linear

$$\begin{aligned} \textbf{maximiza} \quad & z = 4x_1 + x_2 + 3x_3, \\ \textbf{sujeito a} \quad & x_1 + 4x_2 \leq 1, \\ & 3x_1 - x_2 + x_3 \leq 3, \\ & x_1, x_2, x_3 \geq 0. \end{aligned} \tag{3.1}$$

- Cada solução viável fornece um limite inferior para o valor máximo.

$$x_1 = x_2 = x_3 = 0 \Rightarrow z = 0$$
$$x_1 = 1, x_2 = x_3 = 0 \Rightarrow z = 4$$

- Qual a qualidade da solução atual?
- Não sabemos, sem limite superior.

Limites superiores

- Como obter um limite superior?

  Observe: $z = 4x_1 + x_2 + 3x_3 \leq 10x_1 + x_2 + 3x_3 \leq 10$

- Podemos construir uma combinação linear das desigualdades, tal que o coeficiente de cada x_j ultrapasse o coeficiente da função objetivo.

- Nosso exemplo:

$$\begin{aligned} & (x_1 + 4x_2) + 3(3x_1 - x_2 + x_3) \leq 1 + 3 \cdot 3 = 10 \\ \Longleftrightarrow & 10x_1 + x_2 + 3x_3 \leq 10 \end{aligned}$$

- Como obter um limite superior para a função objetivo?

- Qual seria o menor limite superior que esse método fornece?

O menor limite superior

- Sejam $y_1, \ldots, y_n$ os coeficientes de cada linha. Observação: Eles devem ser ≥ 0 para manter a direção das desigualdades.

- Então queremos

$$\begin{aligned} \textbf{minimiza} \quad & \sum_{i \in [m]} b_i y_i \\ \textbf{sujeito a} \quad & \sum_{i \in [m]} a_{ij} y_i \geq c_j, \qquad \forall j \in [n], \\ & y_i \geq 0. \end{aligned}$$

- Isto é o *problema dual* com *variáveis duais* ou *multiplicadores duais* y_i.

Exemplo 3.1
Para o sistema (3.1) obtemos:

$$\begin{aligned} \textbf{minimiza} \quad & y_1 + 3y_2 \\ \textbf{sujeito a} \quad & y_1 + 3y_2 \geq 4, \\ & 4y_1 - y_2 \geq 1, \\ & y_2 \geq 3, \\ & y_1, y_2, y_3 \geq 0. \end{aligned}$$

$\Diamond$

Dualidade: Características

- Em notação matricial

$$\begin{array}{rll} \textbf{maximiza} & c^t x, & \\ \textbf{sujeito a} & Ax \leq b. & \\ & x \geq 0. & \end{array} \qquad \begin{array}{rll} \textbf{minimiza} & b^t y, & \\ \textbf{sujeito a} & y^t A \geq c^t. & \\ & y \geq 0. & \end{array}$$

- O primeiro se chama *primal* e o segundo *dual*.
- Eles usam os mesmos parâmetros c_j, a_{ij}, b_i.

O dual do dual

- Observação: O dual do dual é o primal.
- Forma normal do dual:

$$\begin{array}{rll} -\textbf{maximiza} & -b^t y, & \\ \textbf{sujeito a} & -y^t A \leq -c^t, & \\ & y \geq 0. & \end{array} \quad = \quad \begin{array}{rll} -\textbf{maximiza} & -b^t y, & \\ \textbf{sujeito a} & (-A^t) y \leq -c, & \\ & y \geq 0. & \end{array}$$

- Dual do dual

$$\begin{array}{rll} -\textbf{minimiza} & -c^t z, & \\ \textbf{sujeito a} & z^t(-A^t) \geq -b^t, & \\ & z \geq 0. & \end{array} \quad = \quad \begin{array}{rll} \textbf{maximiza} & c^t z, & \\ \textbf{sujeito a} & Az \leq b, & \\ & z \geq 0. & \end{array}$$

Exemplo 3.2
Qual o dual do problema de transporte (1.11)? Com variáveis duais π_i, $i \in [n]$ para as das restrições de estoque (1.12) e variáveis duais ρ_j, $j \in [m]$ para as restrições de demanda (1.13) obtemos

$$\begin{array}{rll} \textbf{maximiza} & \displaystyle\sum_{i\in[n]} a_i\pi_i + \sum_{j\in[m]} b_j\rho_j, & \\ \textbf{sujeito a} & \pi_i + \rho_j \geq c_{ij}, & \forall i \in [n], j \in [m], \\ & \pi_i, \rho_j \geq 0, & \forall i \in [n], j \in [m]. \end{array} \tag{3.2}$$

$\Diamond$

3.2. Características

Teorema da dualidade fraca

Teorema 3.1 (Dualidade fraca)
Se $x_1, \ldots, x_n$ é uma solução viável do sistema primal, e $y_1, \ldots, y_m$ uma solução viável do sistema dual, então

$$\sum_{i \in [n]} c_i x_i \leq \sum_{j \in [m]} b_j y_j.$$

Prova.

$$\begin{aligned} c^t x &\leq (y^t A)x = y^t(Ax) && \text{pela restrição dual} && (3.3)\\ &\leq y^t b && \text{pela restrição primal} && (3.4) \end{aligned}$$

■

Situação

- Em aberto: Qual o tamanho desse intervalo em geral?

Teorema da dualidade forte

Teorema 3.2
Se $x_1^*, \ldots, x_n^*$ é uma solução ótima do sistema primal, existe uma solução ótima $y_1^*, \ldots, y_m^*$ do sistema dual com

$$\sum_{i \in [n]} c_i x_i^* = \sum_{j \in [m]} b_j y_j^*.$$

Prova. Seja x^* uma solução ótima do sistema primal. Considere um dicionário inicial do método Simplex com variáveis de folga

$$x_{n+j} = b_j - \sum_{i \in [n]} a_{ji} x_i, \qquad \forall j \in [m]$$

e a função objetivo de um dicionário que corresponde com a solução ótima

$$z = z^* + \sum_{i \in [n+m]} \bar{c}_i x_i$$

(com $\bar{c}_i = 0$ para variáveis básicas). Temos que construir uma solução ótima dual y^*. Pela optimalidade, na função objetivo acima, todos $\bar{c}_i$ devem ser não-positivos. Provaremos que $y_j^* = -\bar{c}_{n+j} \geq 0$ para $j \in [m]$ é uma solução dual ótima. Como z^* é o valor ótimo do problema, temos $z^* = \sum_{i \in [n]} c_i x_i^*$. Reescrevendo a função objetivo temos

$$\begin{aligned}
z &= \sum_{i \in [n]} c_i x_i && \text{sistema inicial} \\
&= z^* + \sum_{i \in [n+m]} \bar{c}_i x_i && \text{sistema final} \\
&= z^* + \sum_{i \in [n]} \bar{c}_i x_i + \sum_{j \in [m]} \bar{c}_{n+j} x_{n+j} && \text{separando índices} \\
&= z^* + \sum_{i \in [n]} \bar{c}_i x_i - \sum_{j \in [m]} y_j^* \left(b_j - \sum_{i \in [n]} a_{ji} x_i \right) && \text{subst. solução e var. folga} \\
&= \left(z^* - \sum_{j \in [m]} y_j^* b_j \right) + \sum_{i \in [n]} \left(\bar{c}_i + \sum_{j \in [m]} y_j^* a_{ji} \right) x_i && \text{agrupando}
\end{aligned}$$

Essa derivação está válida para qualquer valor das variáveis x_i, portanto

$$z^* = \sum_{j \in [m]} y_j^* b_j \quad \text{e} \quad c_i = \bar{c}_i + \sum_{j \in [m]} y_j^* a_{ji}, \qquad i \in [n].$$

Logo o primal e dual possuem o mesmo valor

$$\sum_{j \in [m]} y_j^* b_j = z^* = \sum_{i \in [n]} c_i x_i^*$$

e como $\bar{c}_i \leq 0$ sabemos que a solução y^* satisfaz as restrições duais

$$\begin{aligned}
c_i &\leq \sum_{j \in [m]} y_j^* a_{ji}, \qquad i \in [n], \\
y_j^* &\geq 0, \qquad j \in [m].
\end{aligned}$$

■

Consequências: Soluções primais e duais

- Com o teorema da dualidade forte, temos quatro possibilidades

Sistema primal	Sistema dual	Intervalo
Ótimo	Ótimo	Sem
Ilimitado	Inviável	Sem
Inviável	Ilimitado	Sem
Inviável	Inviável	Infinito

Exemplo 3.3 (Primal e dual inviável)
Não segue do teorema da dualidade forte que existe um caso em que tanto o sistema primal quanto o sistema dual são inviáveis. O seguinte exemplo mostra que isso pode acontecer. O sistema primal

$$\begin{aligned} \textbf{maximiza} \quad & x_1 \\ \textbf{sujeito a} \quad & +x_1 - x_2 \leq 0, \\ & -x_1 + x_2 \leq -1, \\ & x_1, x_2 \geq 0, \end{aligned}$$

possui sistema dual correspondente

$$\begin{aligned} \textbf{minimiza} \quad & -y_2 \\ \textbf{sujeito a} \quad & +y_1 - y_2 \geq 1, \\ & -y_1 + y_2 \geq 0. \end{aligned}$$

Ambos os sistemas são inviáveis. ◊

Podemos resumir as possibilidades na seguinte tabela:

	Dual		
Primal	Inviável	Ótimo	Ilimitado
Inviável	√	×	√
Ótimo	×	√	×
Ilimitado	√	×	×

Consequências

- Dado soluções primais e duais x^*, y^* tal que $c^t x^* = b^t y^*$ podemos concluir que ambas soluções são ótimas (x^*, y^* é um *certificado* da optimalidade)[1].
- A prova mostra: com o valor ótimo do sistema primal, sabemos também o valor ótimo do sistema dual.
- Além disso: Podemos trocar livremente entre o sistema primal e dual. $\Rightarrow$ Método Simplex dual.

Outra consequência do Teorema da dualidade forte é o

Teorema 3.3 (Teorema das folgas complementares)
Os vetores x^*, y^* são soluções ótimas do sistema primal e dual, respectivamente, se e somente se

$$y^{*t}(b - Ax^*) = 0 \tag{3.5}$$
$$(y^{*t}A - c^t)x^* = 0 \tag{3.6}$$

Prova. Pelo Teorema da dualidade forte as duas desigualdades (3.3) e (3.4) da prova do Teorema da dualidade fraca se tornam igualdades para soluções ótimas:

$$c^t x^* = y^{*t} A x^* = y^{*t} b$$

Reagrupando termos, o teorema segue. Conversamente, caso (3.5) e (3.6) estão satisfeitos, as soluções primais e duais possuem o mesmo valor e assim tem que ser ótimas. ■

As igualdades 3.5 e 3.6 são ainda válidas em cada componente, porque tanto as soluções ótimas x^*, y^* quanto as folgas primas e duais $b - Ax$ e $y^{*t}A - c^t$ sempre são positivos.

[1] Uma consequência é que o problema de decisão correspondente, determinar se existe uma solução maior que um dado valor, possui um certificado que pode ser verificado em tempo polinomial tanto para uma resposta positiva quanto uma resposta negativa. Portanto, já antes da descoberta de um algoritmo polinomial para esse problema, foi claro que ele pertence a NP ∩ co-NP.

$$x_i > 0 \Rightarrow \sum_{j\in[m]} y_j a_{ji} = c_i \tag{3.7}$$

$$\sum_{j\in[m]} y_j a_{ji} > c_i \Rightarrow x_i = 0 \tag{3.8}$$

$$y_j > 0 \Rightarrow b_j = \sum_{i\in[n]} a_{ji} x_i \tag{3.9}$$

$$b_j > \sum_{i\in[n]} a_{ji} x_i \Rightarrow y_j = 0 \tag{3.10}$$

Como consequência podemos ver que, por exemplo, caso uma igualdade primal não possui folga, a variável dual correspondente é positiva, e, contrariamente, caso uma igualdade primal possui folga, a variável dual correspondente é zero. As mesmas relações se aplicam para as desigualdades no sistema dual. Após a introdução da forma matricial no seção 3.6 vamos analisar a interpretação das variáveis duais com mais detalha no seção 3.7. O teorema das folgas complementares pode ser usado ainda para obter a solução dual dado a solução primal:

Exemplo 3.4
A solução ótima de

$$\begin{aligned} \textbf{maximiza} \quad & z = 6x_1 + 8x_2 + 5x_3 + 9x_4 \\ \textbf{sujeito a} \quad & 2x_1 + x_2 + x_3 + 3x_4 \leq 5, \\ & x_1 + 3x_2 + x_3 + 2x_4 \leq 3, \\ & x_1, x_2, x_3, x_4 \geq 0, \end{aligned}$$

é $x_1 = 2$ e $x_3 = 1$ com valor 17. Pela equação (3.7) sabemos que

$$\begin{aligned} 2y_1 + y_2 &= 6 \\ y_1 + y_2 &= 5. \end{aligned}$$

Portanto a solução dual é $y_1 = 1$ e $y_2 = 4$. ◊

3.3. Dualidade em forma não-padrão

Dualidade em forma padrão

$$\begin{array}{rl} \textbf{maximiza} & c^t x, \\ \textbf{sujeito a} & Ax \leq b, \\ & x \geq 0. \end{array} \qquad \begin{array}{rl} \textbf{minimiza} & b^t y, \\ \textbf{sujeito a} & y^t A \geq c^t, \\ & y \geq 0. \end{array}$$

- O que acontece se o sistema não é em forma padrão?

Igualdades

- Caso de igualdades: Substituindo desigualdades..

$$\begin{array}{rl} \textbf{maximiza} & c^t x, \\ \textbf{sujeito a} & Ax = b, \\ & x \geq 0. \end{array} \qquad \begin{array}{rl} \textbf{maximiza} & c^t x, \\ \textbf{sujeito a} & Ax \leq b, \\ & Ax \geq b, \\ & x \geq 0. \end{array}$$

- ... padronizar novamente, e formar o dual:

$$\begin{array}{rl} \textbf{maximiza} & c^t x, \\ \textbf{sujeito a} & Ax \leq b, \\ & -Ax \leq -b, \\ & x \geq 0. \end{array} \qquad \begin{array}{rl} \textbf{minimiza} & b^t y^+ - b^t y^-, \\ \textbf{sujeito a} & {y^+}^t A - {y^-}^t A \geq c, \\ & y^+ \geq 0, y^- \geq 0, \\ & y^+ = (y_1^+, \ldots, y_m^+)^t, \\ & y^- = (y_1^-, \ldots, y_m^-)^t. \end{array}$$

Igualdades

- Equivalente, usando variáveis irrestritas $y = y^+ - y^-$

$$\begin{array}{rl} \textbf{minimiza} & b^t y \\ \textbf{sujeito a} & y^t A \geq c, \\ & y^t \lesseqgtr 0. \end{array}$$

- Resumo

Primal (max)	Dual (min)
Igualdade	Variável dual livre
Desigualdade ($\leq$)	Variável dual não-negativa
Desigualdade ($\geq$)	Variável dual não-positiva
Variável primal livre	Igualdade
Variável primal não-negativa	Desigualdade ($\geq$)
Variável primal não-positiva	Desigualdade ($\leq$)

Exemplo 3.5 (Exemplo dualidade não-padrão)
O dual de

$$\begin{aligned} \textbf{maximiza} \quad & 3x_1 + x_2 + 4x_3 \\ \textbf{sujeito a} \quad & x_1 + 5x_2 + 9x_3 = 2, \\ & 6x_1 + 5x_2 + 3x_3 \leq 5, \\ & x_1, x_3 \geq 0, x_2 \lesseqgtr 0, \end{aligned}$$

é

$$\begin{aligned} \textbf{minimiza} \quad & 2y_1 + 5y_2 \\ \textbf{sujeito a} \quad & y_1 + 6y_2 \geq 3, \\ & 5y_1 + 5y_2 = 1, \\ & 9y_1 + 3y_2 \geq 4, \\ & y_1 \lesseqgtr 0, y_2 \geq 0. \end{aligned}$$

$\Diamond$

Exemplo 3.6 (Dual do problema de transporte)
O dual do problema de transporte num grafo direcionado $G = (V, A)$ com custos nas arestas c_a, limites inferiores e superiores para o fluxo l_a e u_a em cada arco, e demandas b_v em cada vértice

$$\begin{aligned} \textbf{minimiza} \quad & \sum_{a \in A} c_a x_a \\ \textbf{sujeito a} \quad & \sum_{(u,v) \in A} x_{(u,v)} - \sum_{(v,u) \in A} x_{(v,u)} = b_v, && \forall v \in V, \\ & x_a \geq l_a, && \forall a \in A, \\ & x_a \leq u_a, && \forall a \in A, \\ & x_a \geq 0, && \forall a \in A, \end{aligned}$$

usando variáveis duais $\pi_v \lesseqgtr 0$, $v \in V$, $\rho_a \geq 0$, $a \in A$ e $\sigma_a \leq 0$, $a \in A$ para as três restrições é

$$\begin{aligned}
\textbf{maximiza} \quad & \sum_{v \in V} b_v \pi_v + \sum_{a \in A} l_a \rho_a + u_a \sigma_a & \\
\textbf{sujeito a} \quad & -\pi_u + \pi_v + \rho_a + \sigma_a \geq c_a, & \forall a = (u, v) \in A, \\
& \pi_v \in \mathbb{R}, & \forall v \in V, \\
& \rho_a \geq 0, & \forall a \in A, \\
& \sigma_a \leq 0, & \forall a \in A.
\end{aligned}$$

$\Diamond$

3.4. Interpretação do dual

Exemplo: Dieta dual

- Problema da dieta: Minimiza custos de uma dieta x que alcance dados VDR mínimos.

$$\begin{aligned}
\textbf{minimiza} \quad & c^t x \\
\textbf{sujeito a} \quad & Ax \geq r, \\
& x \geq 0.
\end{aligned}$$

- Unidades das variáveis e parâmetros
  - $x \in \mathbb{R}^n$: Quantidade do alimento $[g]$
  - $c \in \mathbb{R}^n$: R\$/alimento $[R\$/g]$
  - $a_{ij} \in R^{m \times n}$: Nutriente/Alimento $[g/g]$
  - $r \in R^m$: Quantidade de nutriente $[g]$.

Exemplo: Dieta dual

- O problema dual é

$$\begin{aligned}
\textbf{maximiza} \quad & y^t r \\
\textbf{sujeito a} \quad & y^t A \leq c^t, \\
& y \geq 0.
\end{aligned}$$

- Qual a unidade de y? Preço por nutriente $[R\$/g]$.
- Imagine uma empresa, que produz cápsulas que substituem os nutrientes.
- Para vender no mercado, a empresa tem que garantir que uma dieta baseado em cápsulas custa menos que os alimentos correspondentes:

$$\sum_{i\in[m]} y_i a_{ij} \leq c_j, \qquad \forall j \in [m]$$

- Além disso, ela define preços por nutriente que maximizam o custo de uma dieta adequada, para maximizar o próprio lucro.

$$\textbf{maximiza} \quad y^t r$$

Interpretação do dual

- Outra interpretação: o valor de uma variável dual y_j é o *custo marginal* de adicionar mais uma unidade b_j.

  Teorema 3.4
  Se um sistema possui pelo menos uma solução básica ótima não-degenerada, existe um $\epsilon > 0$ tal que, se $|t_j| \leq \epsilon$ para $j \in [m]$,

$$\begin{aligned} \textbf{maximiza} \quad & c^t x \\ \textbf{sujeito a} \quad & Ax \leq b + t, \\ & x \geq 0, \end{aligned}$$

  tem uma solução ótima com valor

$$z = z^* + y^{*t} t$$

  (com z^* o valor ótimo do primal, é y^* a solução ótima do dual).

Exemplo 3.7
Considere uma modificação do sistema do Ildo

$$\begin{aligned} \textbf{maximiza} \quad & 0.2c + 0.5c && (3.11)\\ \textbf{sujeito a} \quad & c + 1.5s \leq 150, && (3.12)\\ & 50c + 50s \leq 6000, && (3.13)\\ & c \leq 80, && (3.14)\\ & s \leq 70, && (3.15)\\ & c, s \geq 0. && (3.16) \end{aligned}$$

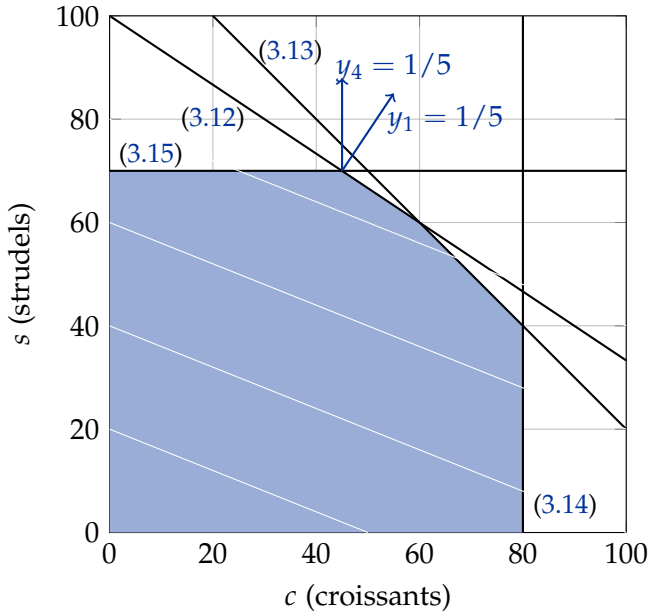


Figura 3.1.: Solução ótima do sistema (3.11) com variáveis duais.

(O sistema foi modificado para a solução ótima atender as condições do teorema 3.4.) A solução ótima do sistema primal é $x^* = (45\,70)^t$ com valor 44, a solução ótima do dual $y^*(1/5\,0\,0\,1/5)^t$. A figura 3.1 mostra a solução ótima com as variáveis duais associadas com as restrições. O valor da variável dual correspondente com uma restrição é o *lucro marginal* de um aumento do lado direito da restrição por um.

$\Diamond$

3.5. Método Simplex dual

Método Simplex dual

- Considere

$$\begin{aligned} \textbf{maximiza} \quad & -x_1 - x_2 \\ \textbf{sujeito a} \quad & -2x_1 - x_2 \leq 4, \\ & -2x_1 + 4x_2 \leq -8, \\ & -x_1 + 3x_2 \leq -7, \\ & x_1, x_2 \geq 0. \end{aligned}$$

- Qual o dual?

$$\begin{aligned} \textbf{minimiza} \quad & 4y_1 - 8y_2 - 7y_3 \\ \textbf{sujeito a} \quad & -2y_1 - 2y_2 - y_3 \geq -1, \\ & -y_1 + 4y_2 + 3y_2 \geq -1, \\ & y_1, y_2, y_3 \geq 0. \end{aligned}$$

Com dicionários

$$\begin{array}{rrrr} z & = & -x_1 & -x_2 \\ \hline w_1 & = 4 & +2x_1 & +x_2 \\ w_2 & = -8 & +2x_1 & -4x_2 \\ w_3 & = -7 & +x_1 & -3x_2 \end{array} \qquad \begin{array}{rrrrr} -w & = & -4y_1 & +8y_2 & +7y_3 \\ \hline z_1 & = 1 & -2y_1 & -2\mathbf{y_2} & -y_3 \\ z_2 & = 1 & -y_1 & +4y_2 & +3y_3 \end{array}$$

- Observação: O primal não é viável, mas o dual é!

- Correspondência das variáveis:

	Variáveis	
	principais	de folga
Primal	$x_1, \ldots, x_n$	$w_1, \ldots, w_m$
Dual	$z_1, \ldots, z_n$, de folga	$y_1, \ldots, y_m$ principais

- Primeiro pivô: y_2 entra, z_1 sai. No primal: w_2 sai, x_1 entra.

Primeiro pivô

$$\begin{array}{llll} z & = -4 & -0.5w_2 & -3x_2 \\ \hline w_1 & = 12 & +w_2 & +5x_2 \\ x_1 & = 4 & +0.5w_2 & +2x_2 \\ w_3 & = -3 & +0.5w_2 & -x_2 \end{array} \qquad \begin{array}{llll} -w & = 4 & -12y_1 & -4z_1 & +3y_3 \\ \hline y_2 & = 0.5 & -y_1 & -0.5z_1 & -0.5\mathbf{y_3} \\ z_2 & = 3 & -5y_1 & -2z_1 & +y_3 \end{array}$$

- Segundo pivô: y_3 entra, y_2 sai. No primal: w_3 sai, w_2 entra.

Segundo pivô

$$\begin{array}{llll} z & = -7 & -w_3 & -4x_2 \\ \hline w_1 & = 18 & +2w_3 & +7x_2 \\ x_1 & = 7 & +w_3 & +3x_2 \\ w_2 & = 6 & +2w_3 & +2x_2 \end{array} \qquad \begin{array}{llll} -w & = 7 & -18y_1 & -7z_1 & -6y_2 \\ \hline y_3 & = 1 & -2y_1 & -z_1 & -2y_2 \\ z_2 & = 4 & -7y_1 & -3z_1 & -2y_2 \end{array}$$

- Sistema dual é ótimo, e portanto o sistema primal também.

Método Simplex dual

- Observação: Não é necessário escrever o sistema dual. Ele é sempre o negativo transposto do sistema primal.

$$z = \bar{z} + \sum_{j \in \mathcal{N}} \bar{c}_j x_j,$$
$$x_i = \bar{b}_i - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j, \qquad i \in \mathcal{B}$$

- Mas é necessário modificar as regras para resolver o sistema dual.

Método Simplex dual: Viabilidade e otimalidade

- Pré-condição: O dicionário é *dualmente viável*, i.e. os coeficientes das variáveis não-básicas na função objetivo tem quer ser não-positivos.

$$\bar{c}_j \leq 0 \quad \text{para} \quad j \in \mathcal{N}.$$

- Otimalidade: Todos variáveis básicas primais positivas

$$\forall i \in \mathcal{B} : \bar{b}_i \geq 0$$

Método Simplex dual: Pivô

- Caso existe uma variável primal negativa: A solução dual não é ótima.
- Regra do maior coeficiente: A variável básica primal de menor valor (que é negativo) sai da base primal.

$$i = \operatorname*{argmin}_{i \in \mathcal{B}} \bar{b}_i$$

- A variável primal nula com fração $\bar{a}_{ij}/\bar{c}_j$ maior entra.

$$j = \operatorname*{argmin}_{\substack{j \in \mathcal{N} \\ \bar{a}_{ij} < 0}} \frac{\bar{c}_j}{\bar{a}_{ij}} = \operatorname*{argmax}_{\substack{j \in \mathcal{N} \\ \bar{a}_{ij} < 0}} \frac{\bar{a}_{ij}}{\bar{c}_j} = \operatorname*{argmax}_{j \in \mathcal{N}} \frac{\bar{a}_{ij}}{\bar{c}_j}$$

Método Simplex dual

Resumo:

- Dualmente viável: $\bar{c}_j \leq 0$ para $j \in \mathcal{N}$.
- Otimalidade: $\forall i \in \mathcal{B} : \bar{b}_i \geq 0$.
- Variável sainte: $i = \operatorname{argmin}_{i \in \mathcal{B}} \bar{b}_i$
- Variável entrante: $j = \operatorname{argmax}_{j \in \mathcal{N}} \frac{\bar{a}_{ij}}{\bar{c}_j}$.

Exemplo

$$\begin{aligned} \textbf{maximiza} \quad & z = -2x_1 - x_2 \\ \textbf{sujeito a} \quad & -x_1 + x_2 \leq -1, \\ & -x_1 - 2x_2 \leq -2, \\ & x_2 \leq 1, \\ & x_1, x_2 \geq 0. \end{aligned}$$

Exemplo: Dicionário inicial

z	$=$	$-2x_1$	$-x_2$
w_1	$=-1$	$+x_1$	$-x_2$
w_2	$=-2$	$+x_1$	$+2\mathbf{x_2}$
w_3	$=1$		$-x_2$

- O dicionário primal não é viável, mas o dual é.

Exemplo: Primeiro pivô

z	$=-1$	$-3/2x_1$	$-1/2w_2$
w_1	$=-2$	$+3/2\mathbf{x_1}$	$-1/2w_2$
x_2	$=1$	$-1/2x_1$	$+1/2w_2$
w_3	$=$	$+1/2x_1$	$-1/2w_2$

Exemplo: Segundo pivô

z	$=-3$	$-w_1$	$-w_2$
x_1	$=4/3$	$+2/3w_1$	$+1/3w_2$
x_2	$=1/3$	$-1/3w_1$	$+1/3w_2$
w_3	$=2/3$	$+1/3w_1$	$-1/3w_2$

3.6. Os métodos em forma matricial

A forma matricial permite uma descrição mais sucinta do método Simplex. A seguir vamos resumir os métodos Simplex primal e dual na forma matricial. Mais importante, nessa forma é possível expressar o dicionário correspondente com qualquer base em termos dos dados inicias (A, c, b). Na próxima seção vamos usar essa forma para analisar a sensibilidade de uma solução à pequenas perturbações dos dados (i.e. os coeficientes A,b, e c).

3.6.1. O dicionário final em função dos dados

Sistema padrão

- O sistema padrão é

$$\begin{aligned} \textbf{maximiza} \quad & c^t x \\ \textbf{sujeito a} \quad & Ax \leq b, \\ & x \geq 0. \end{aligned}$$

- Com variáveis de folga $x_{n+1}, \ldots, x_{n+m}$ e *A,c,x novo* (definição segue abaixo)

$$\begin{aligned} \textbf{maximiza} \quad & c^t x \\ \textbf{sujeito a} \quad & Ax = b, \\ & x \geq 0. \end{aligned}$$

Matrizes

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} & 1 & & \\ a_{21} & a_{22} & \cdots & a_{2n} & & 1 & \\ \vdots & \vdots & & \vdots & & & \ddots \\ a_{m1} & a_{m2} & \ldots & a_{mn} & & & 1 \end{pmatrix};$$

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}; c = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \\ 0 \\ \vdots \\ 0 \end{pmatrix}; x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ x_{n+1} \\ \vdots \\ x_{n+m} \end{pmatrix}$$

Separação das variáveis

- Em cada iteração as variáveis estão separados em básicas e não-básicas.
- Conjuntos de índices correspondentes: $\mathcal{B} \dot{\cup} \mathcal{N} = [1, n+m]$.
- A componente i de Ax pode ser separado como

$$\sum_{j \in [n+m]} a_{ij} x_j = \sum_{j \in \mathcal{B}} a_{ij} x_j + \sum_{j \in \mathcal{N}} a_{ij} x_j.$$

Separação das variáveis

- Para obter a mesma separação na forma matricial: Reordenamos as colunas e separamos as matrizes e vetores:

$$A = (B\,N)\,; \quad x = \begin{pmatrix} x_B \\ x_N \end{pmatrix}; \quad c = \begin{pmatrix} c_B \\ c_N \end{pmatrix}$$

- com $B \in \mathbb{R}^{m\times m}$, $N \in \mathbb{R}^{m\times n}$, $c \in \mathbb{R}^{n+m}$.

Forma matricial das equações

- Agora $Ax = b$ é equivalente com

$$(B\,N)\begin{pmatrix} x_B \\ x_N \end{pmatrix} = Bx_B + Nx_N = b$$

- Numa solução básica, a matriz B tem posto m tal que as colunas de B formam uma *base* do $\mathbb{R}^m$. Logo B possui inversa e

$$x_B = B^{-1}(b - Nx_N) = B^{-1}b - B^{-1}Nx_N$$

Forma matricial da função objetivo

- A função objetivo é

$$z = c^t x = (c_B^t\, c_N^t)\begin{pmatrix} x_B \\ x_N \end{pmatrix} = c_B^t x_B + c_N^t x_N$$

- e usando $x_B = B^{-1}b - B^{-1}Nx_N$ obtemos

$$\begin{aligned} z &= c_B^t(B^{-1}b - B^{-1}Nx_N) + c_N^t x_N \\ &= c_B^t B^{-1}b - (c_B^t B^{-1}N - c_N^t)x_N \\ &= c_B^t B^{-1}b - ((B^{-1}N)^t c_B - c_N)^t x_N \end{aligned}$$

Dicionário em forma matricial

- Logo, o dicionário em forma matricial é

$$\frac{z = c_B^t B^{-1} b - ((B^{-1}N)^t c_B - c_N)^t x_N}{x_B = B^{-1} b - B^{-1} N x_N}$$

- Compare com a forma em componentes:

$$z = \bar{z} + \sum_{j \in \mathcal{N}} \bar{c}_j x_j \qquad\qquad z = \bar{z} + \bar{c}^t x_N$$
$$x_i = \bar{b}_i - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j \qquad i \in \mathcal{B} \qquad\qquad x_B = \bar{b} - \bar{A} x_N$$

Dicionário em forma matricial

- Portanto, vamos identificar

$$\bar{z} = c_B^t B^{-1} b; \qquad \bar{c} = -((B^{-1}N)^t c_B - c_N)$$
$$\bar{b} = B^{-1} b; \qquad \bar{A} = (\bar{a}_{ij}) = B^{-1} N$$

- para obter o dicionário

$$z = \bar{z} + \bar{c}^t x_N$$
$$x_B = \bar{b} - \bar{A} x_N$$

Sistema dual

- As variáveis primais são

$$x = (\underbrace{x_1 \dots x_n}_{\text{original}} \underbrace{x_{n+1} \dots x_{n+m}}_{\text{folga}})^t$$

- Para manter índices correspondentes, escolhemos variáveis duais da forma

$$y = (\underbrace{y_1 \dots y_n}_{\text{folga}} \underbrace{y_{n+1} \dots y_{n+m}}_{\text{dual}})^t$$

- O dicionário do dual correspondente então é

Primal	Dual
$z = \bar{z} + \bar{c}^t x_N$	$-w = -\bar{z} - \bar{b}^t y_B$
$x_B = \bar{b} - \bar{A} x_N$	$y_N = -\bar{c} + \bar{A}^t y_B$

Primal e dual

- A solução básica do sistema primal é

$$x_N^* = 0; \quad x_B^* = \bar{b} = B^{-1}b$$

- A solução dual correspondente é

$$y_B^* = 0; \quad y_N^* = -\bar{c} = (B^{-1}N)^t c_B - c_N$$

- Com isso temos os dicionários

$$\begin{aligned} z &= \bar{z} - (y_N^*)^t x_N & -w &= -\bar{z} - (x_B^*)^t y_B \\ x_B &= x_B^* - (B^{-1}N)x_N & y_N &= y_N^* + (B^{-1}N)^t y_B \end{aligned}$$

Observação 3.1
A solução dual completa é $y^t = c_B^t B^{-1} A - c^t$ (isso pode ser visto como?), ou $y_i = c_B^t B^{-1} a^i - c_i$ para cada índice $i \in [n+m]$. As variáveis duais originais com índice $i \in [n+1, m]$ correspondem com as colunas $a^i = e_i$ das variáveis de folga e possuem coeficientes $c_i = 0$. Logo $y_o^t = c_B^t B^{-1}$ é a solução do sistema dual sem as variáveis de folga, e podemos escrever $y = (y_o^t A - c^t)^t = A^t y_o - c$ e para os custos reduzidos $\bar{c} = c - A^t y_o$. ◊

3.6.2. Simplex em forma matricial

Método Simplex em forma matricial

- Começamos com uma partição $\mathcal{B} \dot{\cup} \mathcal{N} = [1, n+m]$.
- Em cada iteração selecionamos uma variável sainte $i \in \mathcal{B}$ e entrante $j \in \mathcal{N}$.
- Fazemos o pivô x_i com x_j.
- Depois a nova base é $\mathcal{B} \setminus \{i\} \cup \{j\}$.

Método Simplex em forma matricial

S1: Verifique solução ótima Se $y_N^* \geq 0$ a solução atual é ótima. Pare.

S2: Escolhe variável entrante Escolhe $j \in \mathcal{N}$ com $y_j^* < 0$. A variável entrante é x_j.

S3: Determine passo básico Aumentando x_j uma unidade temos novas variáveis não-básicas $x_N = x_N^* + \Delta x_N$ com $\Delta x_N = (0 \cdots 010 \cdots 0)^t = e_j$ e e_j o vetor nulo com somente 1 na posição correspondente com índice j. Como

$$x_B = x_B^* - B^{-1}Nx_N,$$

a diminuição correspondente das variáveis básicas é $\Delta x_B = B^{-1}Ne_j$.

Método Simplex em forma matricial

S4: Determine aumento máximo O aumento máximo de x_j é limitado por $x_B \geq 0$, i.e.

$$x_B = x_B^* - t\Delta x_B \geq 0 \Longleftrightarrow x_B^* \geq t\Delta x_B.$$

Com $t, x_B^* \geq 0$ temos

$$t \leq t^* = \min_{\substack{i \in \mathcal{B} \\ \Delta x_i > 0}} \frac{x_i^*}{\Delta x_i}$$

S5: Escolhe variável sainte Escolhe um $i \in \mathcal{B}$ com $x_i^* = t^* \Delta x_i$.

Método Simplex em forma matricial

S5: Determine passo dual A variável entrante dual é y_i. Aumentando uma unidade, as variáveis y_N diminuem $\Delta y_N = -(B^{-1}N)^t e_i$.

S6: Determina aumento máximo Com variável sainte y_j, sabemos que y_i pode aumentar ao máximo

$$s = \frac{y_j^*}{\Delta y_j}.$$

S7: Atualiza solução

$$\begin{aligned} x_j^* &:= t & y_i^* &:= s \\ x_B^* &:= x_B^* - t\Delta x_B & y_N^* &:= y_N^* - s\Delta y_N \\ \mathcal{B} &:= \mathcal{B} \setminus \{i\} \cup \{j\} \end{aligned}$$

3.7. Análise de sensibilidade

Motivação

- Na solução da programas lineares os parâmetros são fixos.
- Qual o efeito de uma perturbação

$$c := c + \Delta c; \quad b := b + \Delta b; \quad A := A + \Delta A?$$

(Imagina erros de medida, pequenas flutuações, etc.)

Análise de sensibilidade

- Após a solução de um sistema linear, temos o dicionário ótimo

$$z = z^* - (y_N^*)^t x_N$$
$$x_B = x_B^* - B^{-1} N x_N$$

- com

$$x_B^* = B^{-1} b$$
$$y_N^* = (B^{-1} N)^t c_B - c_N$$
$$z^* = c_B^t B^{-1} b$$

Modificar c

- Mudarmos c para $\hat{c}$, mantendo a base $\mathcal{B}$.
- x_B^* não muda, mas temos que reavaliar y_N^* e z^*.
- Depois, x_B^* ainda é uma solução básica viável do sistema primal.
- Logo, podemos continuar aplicando o método Simplex primal.

Modificar b

- Da mesma forma, modificamos b para $\hat{b}$ (mantendo a base).
- y_N^* não muda, mas temos que reavaliar x_B^* e z^*.
- Depois, y_N^* ainda é uma solução básica viável do sistema dual.
- Logo, podemos continuar aplicando o método Simplex dual.

Vantagem dessa abordagem

- Nos dois casos, esperamos que a solução inicial já é perto da solução ótima.
- Experiência prática confirma isso.
- O que acontece se queremos modificar tanto b quanto c ou ainda A?
- A solução atual não necessariamente é viável no sistema primal ou dual.
- Mas: Mesmo assim, a convergência na prática é mais rápido.

Estimar intervalos

- Pergunta estendida: Qual o intervalo de $t \in R$ tal que o sistema com $\hat{c} = c + t\Delta c$ permanece ótimo?
- Para $t = 1$: $y_N^* = (B^{-1}N)^t c_B - c_N$ aumenta $\Delta y_N := (B^{-1}N)^t \Delta c_B - \Delta c_N$.
- Em geral: Aumento $t\Delta y_N$.
- Condição para manter a viabilidade dual:

$$y_N^* + t\Delta y_N \geq 0$$

- Para $t > 0$ temos

$$t \leq \min_{\substack{j \in \mathcal{N} \\ \Delta y_j < 0}} -\frac{y_j^*}{\Delta y_j}$$

- Para $t < 0$ temos

$$\max_{\substack{j \in \mathcal{N} \\ \Delta y_j > 0}} -\frac{y_j^*}{\Delta y_j} \leq t$$

Estimar intervalos

- Agora seja $\hat{b} = b + t\Delta b$.
- Para $t = 1$: $x_B^* = B^{-1}b$ aumenta $\Delta x_B := B^{-1}\Delta b$.
- Em geral: Aumento $t\Delta x_B$.

- Condição para manter a viabilidade primal:

$$x_B^* + t\Delta x_B \geq 0$$

- Para $t > 0$ temos

$$t \leq \min_{\substack{i \in \mathcal{B} \\ \Delta x_i < 0}} -\frac{x_i^*}{\Delta x_i}$$

- Para $t < 0$ temos

$$\max_{\substack{i \in \mathcal{B} \\ \Delta x_i > 0}} -\frac{x_i^*}{\Delta x_i} \leq t$$

Observação 3.2
A matriz B^{-1} é formada pelas colunas do dicionário final que correspondem com as variáveis de folga. ◊

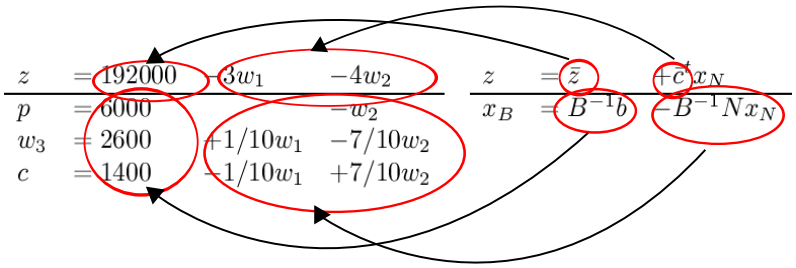
Exemplo 3.8
Considere o problema da empresa de aço (visto na aula prática, veja também execício 1.7).

$$\begin{aligned} \textbf{maximiza} \quad & 25p + 30c \\ \textbf{sujeito a} \quad & 7p + 10c \leq 56000, \\ & p \leq 6000, \\ & c \leq 4000, \\ & p, c \geq 0. \end{aligned}$$

Qual o intervalo em que o valor do lucro das placas de $25R\$$ pode variar sem alterar a solução ótima?

Exemplo: Empresa de aço

- Sistema ótimo



- Base $\mathcal{B} = \{p, w_3, c\}$, variáveis não-básicas $\mathcal{N} = \{w_1, w_2\}$. (Observe: usamos conjuntos de variáveis, ao invés de conjuntos de índices).

Exemplo: Variáveis

- Vetores c e Δc. Observe que reordenamos os dados do sistema inicial de forma correspondente com a ordem das variáveis do sistema final.

$$c = \begin{pmatrix} 25 \\ 0 \\ 30 \\ 0 \\ 0 \end{pmatrix}; c_B = \begin{pmatrix} 25 \\ 0 \\ 30 \end{pmatrix}; c_N = \begin{pmatrix} 0 \\ 0 \end{pmatrix};$$

$$\Delta c = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}; \Delta c_B = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}; \Delta c_N = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Exemplo: Aumentos

- Aumento das variáveis duais

$$\Delta y_N = (B^{-1}N)^t \Delta c_B - \Delta c_N = (B^{-1}N)^t \Delta c_B$$

- com

$$B^{-1}N = \begin{pmatrix} 0 & 1 \\ -1/10 & 7/10 \\ 1/10 & -7/10 \end{pmatrix}$$

- temos

$$\Delta y_N = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Exemplo: Limites

- Limites em geral

$$\max_{\substack{j \in \mathcal{N} \\ \Delta y_j > 0}} -\frac{y_j^*}{\Delta y_j} \leq t \leq \min_{\substack{j \in \mathcal{N} \\ \Delta y_j < 0}} -\frac{y_j^*}{\Delta y_j}$$

- Logo
$$-4 \leq t \leq \infty.$$
- Uma variação do preço entre $25 + [-4, \infty] = [21, \infty]$ preserve a otimalidade da solução atual.
- O novo valor da função objetivo é
$$z == \hat{c}_B^t B^{-1} b = \begin{pmatrix} 25+t & 0 & 30 \end{pmatrix} \begin{pmatrix} 6000 \\ 2600 \\ 1400 \end{pmatrix} = 192000 + 6000t$$
e os valores das variáveis p e c permanecem os mesmos.

$\Diamond$

Exemplo 3.9
Qual o intervalo em que o lucro das placas (R$ 25) e dos canos (R$ 30) podem variar sem que a solução ótima seja alterada?

Exemplo: Variação do lucro dos placas e canos

- Os vetores c, c_B, c_N e Δc_N permanecem os mesmos do exemplo anterior. Enquanto que:
$$\Delta c = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}; \Delta c_B = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix};$$
- Neste caso, o valor de Δy_N é
$$\Delta y_N = (B^{-1}N)^t \Delta c_B = \begin{pmatrix} 0 & -1/10 & 1/10 \\ 1 & 7/10 & -7/10 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1/10 \\ 3/10 \end{pmatrix}.$$
- Logo $-40/3 \leq t \leq \infty$
- Ou seja, uma variação do lucro das placas entre R$ 11.67 e ∞, e do lucro dos canos entre R$ 16.67 e ∞, não altera a solução ótima do sistema.

$\Diamond$

Exemplo: Modificação

- Qual o intervalo em que o lucro dos canos (R$ 30) podem variar sem que a solução ótima seja alterada?
- Os vetores c, c_B, c_N e Δc_N permanecem os mesmos do exemplo anterior. Enquanto que:

$$\Delta c = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}; \Delta c_B = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix};$$

- Neste caso, o valor de Δy_N é:

$$\Delta c_B = \begin{pmatrix} 1/10 \\ -7/10 \end{pmatrix};$$

- Logo $-30 \leq t \leq 40/7$
- Ou seja, uma variação do lucro dos canos entre R$ 0 e R$ 35.71, não altera a solução ótima do sistema.

Exemplo 3.10
O que acontece se mudarmos o lucro das placas para R$ 20?

Exemplo: Placas com lucro R$ 20

- Novos vetores

$$\hat{c} = \begin{pmatrix} 20 \\ 0 \\ 30 \\ 0 \\ 0 \end{pmatrix}; \hat{c}_B = \begin{pmatrix} 20 \\ 0 \\ 30 \end{pmatrix}; \hat{c}_N = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

- Aumento

$$\begin{aligned} \hat{y}_N^* &= (B^{-1}N)^t \hat{c}_B - \hat{c}_N = (B^{-1}N)^t \hat{c}_B \\ &= \begin{pmatrix} 0 & -1/10 & 1/10 \\ 1 & 7/10 & -7/10 \end{pmatrix} \begin{pmatrix} 20 \\ 0 \\ 30 \end{pmatrix} = \begin{pmatrix} 3 \\ -1 \end{pmatrix} \end{aligned}$$

Novas variáveis

- Com
$$B^{-1}b = \begin{pmatrix} 6000 \\ 2600 \\ 1400 \end{pmatrix}$$
- Novo valor da função objetivo
$$\hat{z}^* = \hat{c}_B^t B^{-1} b = \begin{pmatrix} 20 & 0 & 30 \end{pmatrix} \begin{pmatrix} 6000 \\ 2600 \\ 1400 \end{pmatrix} = 162000$$

Exemplo: Novo dicionário

- Novo sistema primal viável, mas não ótimo:

z	$= 162000$	$-3w_1$	$+w_2$
p	$= 6000$		$-w_2$
w_3	$= 2600$	$+1/10w_1$	$-7/10\mathbf{w_2}$
c	$= 1400$	$-1/10w_1$	$+7/10w_2$

- Depois um pivô: Sistema ótimo.

z	$= 165714\,2/7$	$-20/7w_1$	$-10/7w_3$
p	$= 2285\,5/7$	$-1/7w_1$	$+10/7w_3$
w_2	$= 3714\,2/7$	$+1/7w_1$	$-10/7w_2$
c	$= 4000$		$-w_3$

$\Diamond$

Exemplo 3.11
O que acontece se mudarmos o lucro das placas de R$ 25 para R$ 35 e dos canos de R$ 30 para R$ 10?

Exemplo: Placas e canos com lucro R$ 35 e R$ 10

- Novos vetores
$$\hat{c} = \begin{pmatrix} 35 \\ 0 \\ 10 \\ 0 \\ 0 \end{pmatrix}; \quad \hat{c}_B = \begin{pmatrix} 35 \\ 0 \\ 10 \end{pmatrix}; \quad \hat{c}_N = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

- Aumento

$$\hat{y}_N^* = ((B^{-1}N)^t c_B - c_N) = \begin{pmatrix} 0 & -1/10 & 1/10 \\ 1 & 7/10 & -7/10 \end{pmatrix} \begin{pmatrix} 35 \\ 0 \\ 10 \end{pmatrix} = \begin{pmatrix} 1 \\ 28 \end{pmatrix}$$

Novas variáveis e novo dicionário

- Novo valor da função objetivo

$$\hat{z}^* = \hat{c}_B^t B^{-1} b = \hat{c}_B^t x_B^* = \begin{pmatrix} 35 & 0 & 10 \end{pmatrix} \begin{pmatrix} 6000 \\ 2600 \\ 1400 \end{pmatrix} = 224000$$

- O novo sistema primal viável é

z	$= 224000$	$-1w_1$	$-28w_2$
p	$= 6000$		$-w_2$
w_3	$= 2600$	$+1/10w_1$	$-7/10w_2$
c	$= 1400$	$-1/10w_1$	$+7/10w_2$

- O sistema é ótimo.

$\Diamond$

Exemplo 3.12
Qual o efeito de uma variação do lado direito 6000 da segunda restrição? Para estudar essa variação escolhemos $\Delta b = (0\,1\,0)^t$. Temos, pela Observção 3.2

$$B^{-1} = 1/10 \begin{pmatrix} 0 & 10 & 0 \\ -1 & 7 & 10 \\ 1 & -7 & 0 \end{pmatrix}$$

e logo $\Delta x_B = B^{-1}\Delta b = 1/10(10\,7\,-7)^t$. Obtemos a nova solução básica

$$\hat{x}_B^* = \begin{pmatrix} 6000 \\ 2600 \\ 1400 \end{pmatrix} + t/10 \begin{pmatrix} 10 \\ 7 \\ -7 \end{pmatrix}$$

e a condição de otimalidade $\hat{x}_B^* \geq 0$ nos fornece os limites

$$-26000/7 \leq t \leq 2000$$

entre quais ela é ótima. O valor da função objetivo dentro desses limites é

$$\hat{z}^* = c_B^t \hat{x}_B^* = (25\,0\,30)^t \begin{pmatrix} 6000+t \\ 2600+7/10t \\ 1400-7/10t \end{pmatrix} = 192000+4t.$$

◊

3.8. Exercícios

(Soluções a partir da página 220.)

Exercício 3.1
Qual o sistema dual de

$$\begin{aligned} \textbf{minimiza} \quad & 7x_1 + x_2 + 5x_3 \\ \textbf{sujeito a} \quad & x_1 - x_2 + 3x_3 \geq 10, \\ & 5x_1 + 2x_2 - x_3 \geq 6, \\ & x_1, x_2, x_3 \geq 0? \end{aligned}$$

Exercício 3.2
Considere o problema

COBERTURA POR CONJUNTOS PONDERADOS (WEIGHTED SET COVER)

Instância Um universo U, uma familia $\mathcal{S}$ de subconjuntos do universo, i.e. para todo $S \in \mathcal{S}$, $S \subseteq U$, e custos $c(S)$ para cada conjunto $S \in \mathcal{S}$.

Solução Uma cobertura por conjuntos, i.e. uma seleção de conjuntos $\mathcal{T} \subseteq \mathcal{S}$ tal que para cada elemento $e \in U$ existe pelo menos um $S \in \mathcal{T}$ com $e \in S$.

Objetivo Minimizar o custo total dos conjuntos selecionados.

Uma formulação inteira do problema é

$$\begin{aligned} \textbf{minimiza} \quad & \sum_{S \in \mathcal{S}} c(S) x_S \\ \textbf{sujeito a} \quad & \sum_{S : e \in S} x_S \geq 1, && e \in U, \\ & x_S \in \mathbb{B}, && S \in \mathcal{S}. \end{aligned}$$

O problema com restrições de integralidade é NP-completo. Substituindo as restrições de integralidade $x_S \in \mathbb{B}$ por restrições triviais $x_S \geq 0$ obtemos um programa linear. Qual o seu dual?

Exercício 3.3
O sistema

$$\begin{aligned} \textbf{maximiza} \quad & 2x_1 - x_2 + x_3 \\ \textbf{sujeito a} \quad & 3x_1 + x_2 + x_3 \leq 60, \\ & x_1 - x_2 + 2x_3 \leq 10, \\ & x_1 + x_2 - x_3 \leq 20, \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

possui dicionário ótimo

$$\begin{array}{rrrrr} z = & 25 & -3/2x_5 & -1/2x_6 & -3/2x_3 \\ \hline x_4 = & 10 & +x_5 & +2x_6 & -x_3 \\ x_1 = & 15 & -1/2x_5 & -1/2x_6 & -1/2x_3 \\ x_2 = & 5 & +1/2x_5 & -1/2x_6 & +3/2x_3 \end{array}$$

a) Em qual intervalo o coeficiente $c_1 = 2$ pode variar?

b) Em qual intervalo o coeficiente $b_2 = 10$ pode variar?

c) Modifique o lado direito de $(60\ 10\ 20)^t$ para $(70\ 20\ 10)^t$: o sistema mantém-se ótimo? Caso contrário, determina a nova solução ótima.

d) Modifique a função objetivo para $3x_1 - 2x_2 + 3x_3$: o sistema mantém-se ótimo? Caso contrário, determina a nova solução ótima.

4. Tópicos

4.1. Centro de Chebyshev

Seja $B(c,r) = \{c+u \mid ||u|| \leq r\}$ a esfera com centro c e raio r. Para um polígono convexo $a_i x \leq b_i$, para $i \in [n]$, queremos encontrar o centro e o raio da maior esfera, que cabe dentro do polígono, i.e. resolver

$$\begin{aligned} \textbf{maximiza} \quad & r \\ \textbf{sujeito a} \quad & \sup_{p \in B(c,r)} a_i p \leq b_i, \qquad \forall i \in [n]. \end{aligned}$$

Temos

$$\sup_{p \in B(c,r)} a_i p = c a_i + \sup_{||u|| \leq r} a_i u = c a_i + ||a_i|| r$$

porque o segundo supremo é atingido por $u = r a_i / ||a_i||$. Assim obtemos uma formulação linear

$$\begin{aligned} \textbf{maximiza} \quad & r \\ \textbf{sujeito a} \quad & a_i c + r||a_i|| \leq b_i, \qquad \forall i \in [n]. \end{aligned}$$

Exemplo 4.1
O polígono da Fig. 4.1 possui a descrição

$$\begin{aligned} 2x_1 + 4x_2 &\leq 24, \\ 4x_1 - x_2 &\leq 12, \\ -x_1 &\leq 0, \\ -x_2 &\leq 0. \end{aligned}$$

Portanto o programa linear para encontrar o centro e o raio do maior círculo é

$$\begin{aligned} \textbf{maximiza} \quad & r \\ \textbf{sujeito a} \quad & 2c_1 + 4c_2 + \sqrt{20} r \leq 24, \\ & 4c_1 - c_2 + \sqrt{17} r \leq 12, \\ & -c_1 + r \leq 0, \\ & -c_2 + r \leq 0. \end{aligned}$$

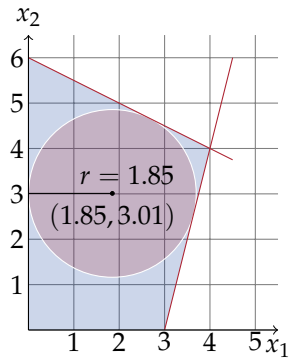


Figura 4.1.: Exemplo do centro de Chebyshev

◊

4.2. Função objetivo convexa e linear por segmentos

Uma função f é *convexa* se $f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$ para qualquer x e y e $0 \leq t \leq t$. Funções convexas são importantes na otimização, porque eles possuem no máximo um mínimo no interior do domínio deles, e portanto o mínimo de uma função convexa pode ser obtido com métodos locais.
Seja $f_i(x), i \in [n]$ uma coleção de funções lineares. O máximo $f(x) = \max_{i\in[n]} f_i(x)$ é uma função convexa linear por segmentos. O problema de otimização

$$\textbf{minimiza} \max_{i\in[n]} f_i(x)$$

é equivalente com o programa linear

$$\begin{aligned} \textbf{minimiza} \quad & x_0 && (4.1)\\ \textbf{sujeito a} \quad & f_i(x) \leq x_0, \qquad \forall i \in [n]. && (4.2) \end{aligned}$$

Portanto podemos minimizar uma função convexa linear por segmentos usando programação linear. De forma similar, f é *concava* se $f(tx + (1-t)y) \geq tf(x) + (1-t)f(y)$. (Observe que uma função convexa e concava é afina.) O sistema

$$\begin{aligned} \textbf{maximiza} \quad & x_0 \\ \textbf{sujeito a} \quad & f_i(x) \geq x_0, \\ & x\forall i \in [n]. \end{aligned}$$

maximiza uma função concava linear por segmentos.

Parte II.

Programação inteira

5. Introdução

5.1. Definições

Problema da dieta

- Problema da dieta

$$\begin{aligned} \textbf{minimiza} \quad & c^t x \\ \textbf{sujeito a} \quad & Ax \geq r, \\ & x \geq 0. \end{aligned}$$

- Uma solução (laboratório): 5 McDuplos, 3 maçãs, 2 casquinhas mista para R$ 24.31
- Mentira! Solução correta: 5.05 McDuplos, 3.21 maças, 2.29 casquinhas mistas.
- Observação: Correto somente em média sobre várias refeições.

Como resolver?

- Com saber o valor ótima para uma única refeição?
- Restringe as variáveis x ao conjunto $\mathbb{Z}$.
- Será que método Simplex ainda funciona?
- Não. Pior: O problema torna-se NP-completo.

Problemas de otimização

- Forma geral

$$\begin{aligned} \textbf{optimiza} \quad & f(x), \\ \textbf{sujeito a} \quad & x \in V. \end{aligned}$$

Programação inteira

- Programação linear (PL)

$$\begin{aligned} \textbf{maximiza} \quad & c^t x \\ \textbf{sujeito a} \quad & Ax \leq b, \\ & x \in \mathbb{R}^n \geq 0; \end{aligned}$$

- Programação inteira pura (PI)

$$\begin{aligned} \textbf{maximiza} \quad & h^t y \\ \textbf{sujeito a} \quad & Gy \leq b, \\ & y \in \mathbb{Z}^n \geq 0. \end{aligned}$$

Programação inteira

- Programação (inteira) mista (PIM)

$$\begin{aligned} \textbf{maximiza} \quad & c^t x + h^t y \\ \textbf{sujeito a} \quad & Ax + Gy \leq b, \\ & x \in \mathbb{R}^n \geq 0, y \in \mathbb{Z}^m \geq 0; \end{aligned}$$

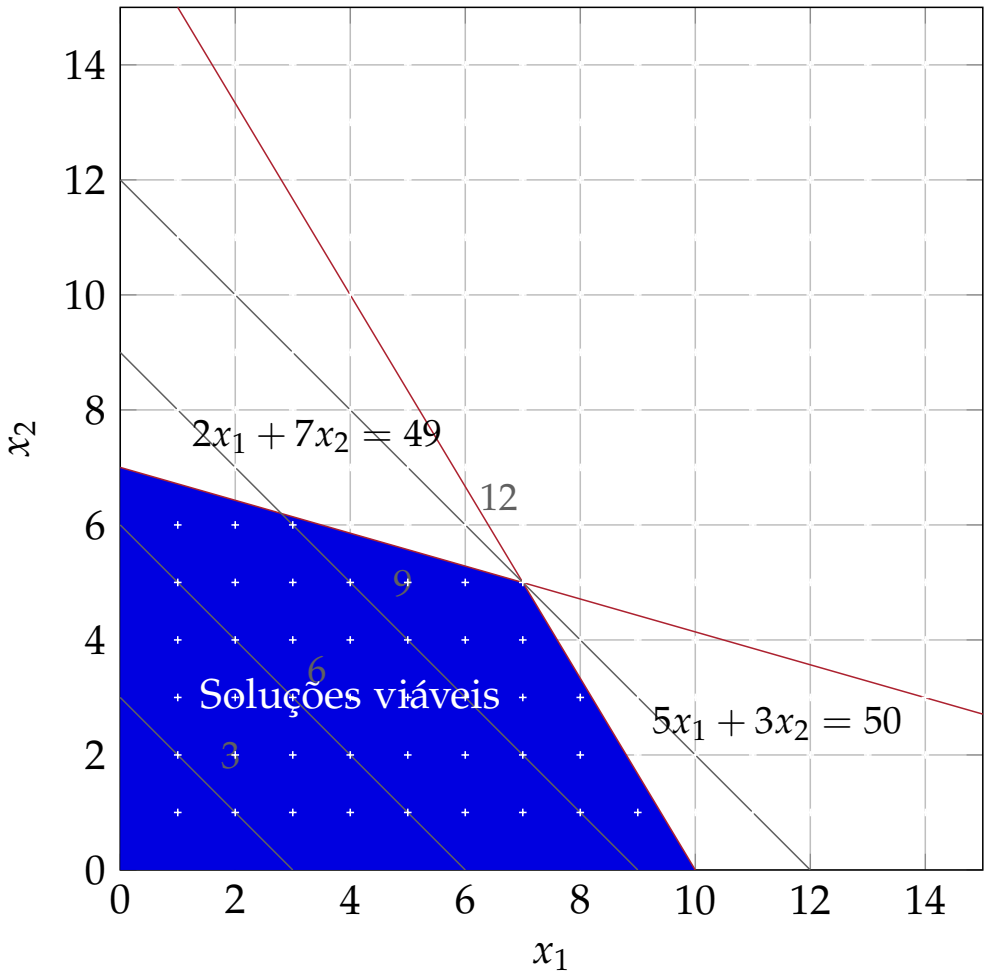
- Programação linear e inteira pura são casos particulares da programação mista.
- Outro caso particular: 0-1-PIM e 0-1-PI.

$$x \in \mathbb{B}^n$$

Exemplo

$$\begin{aligned} \textbf{maximiza} \quad & x_1 + x_2 \\ \textbf{sujeito a} \quad & 2x_1 + 7x_2 \leq 49, \\ & 5x_1 + 3x_2 \leq 50. \end{aligned}$$

Exemplo



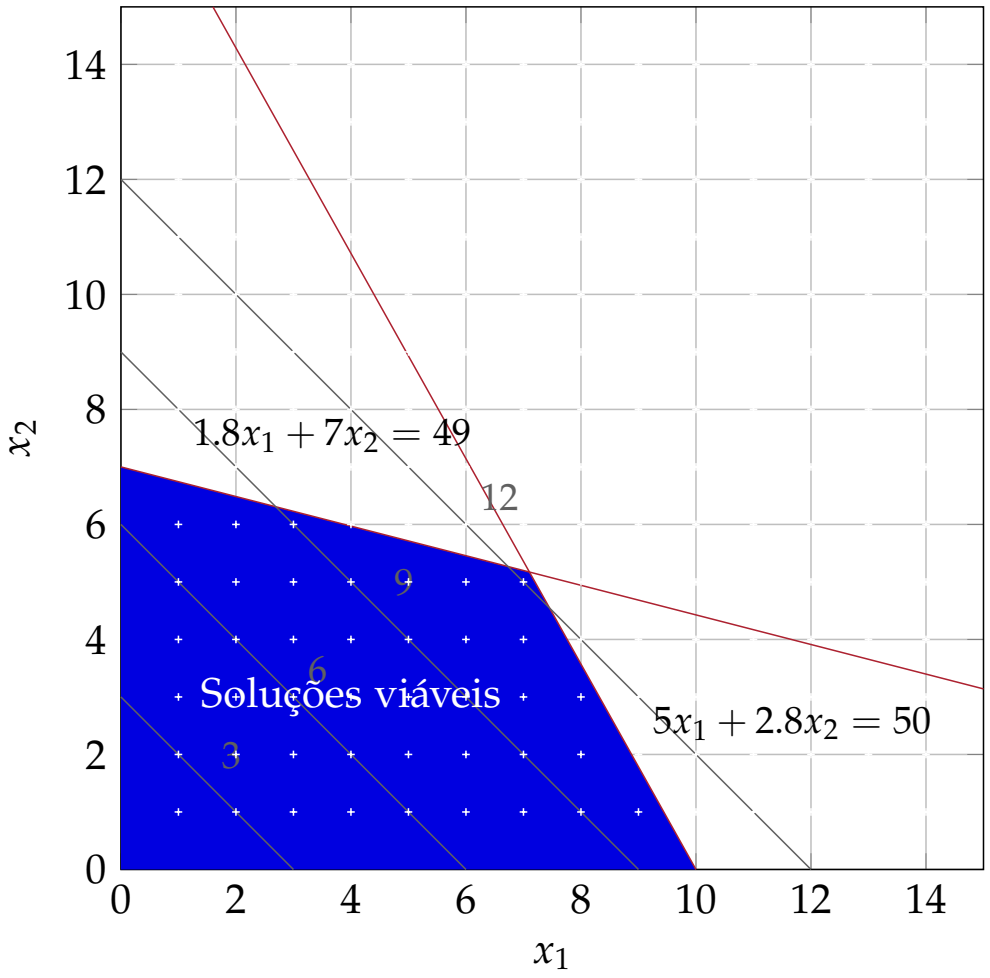
- Sorte: A solução ótima é inteira! $x_1 = 7$, $x_2 = 5$, $V = 12$.

- Observação: Se a solução ótima é inteira, um problema de PI(M) pode ser resolvido com o método Simplex.

Exemplo

$$\begin{aligned} \textbf{maximiza} \quad & x_1 + x_2 \\ \textbf{sujeito a} \quad & \mathbf{1.8}x_1 + 7x_2 \leq 49, \\ & 5x_1 + \mathbf{2.8}x_2 \leq 50. \end{aligned}$$

Exemplo



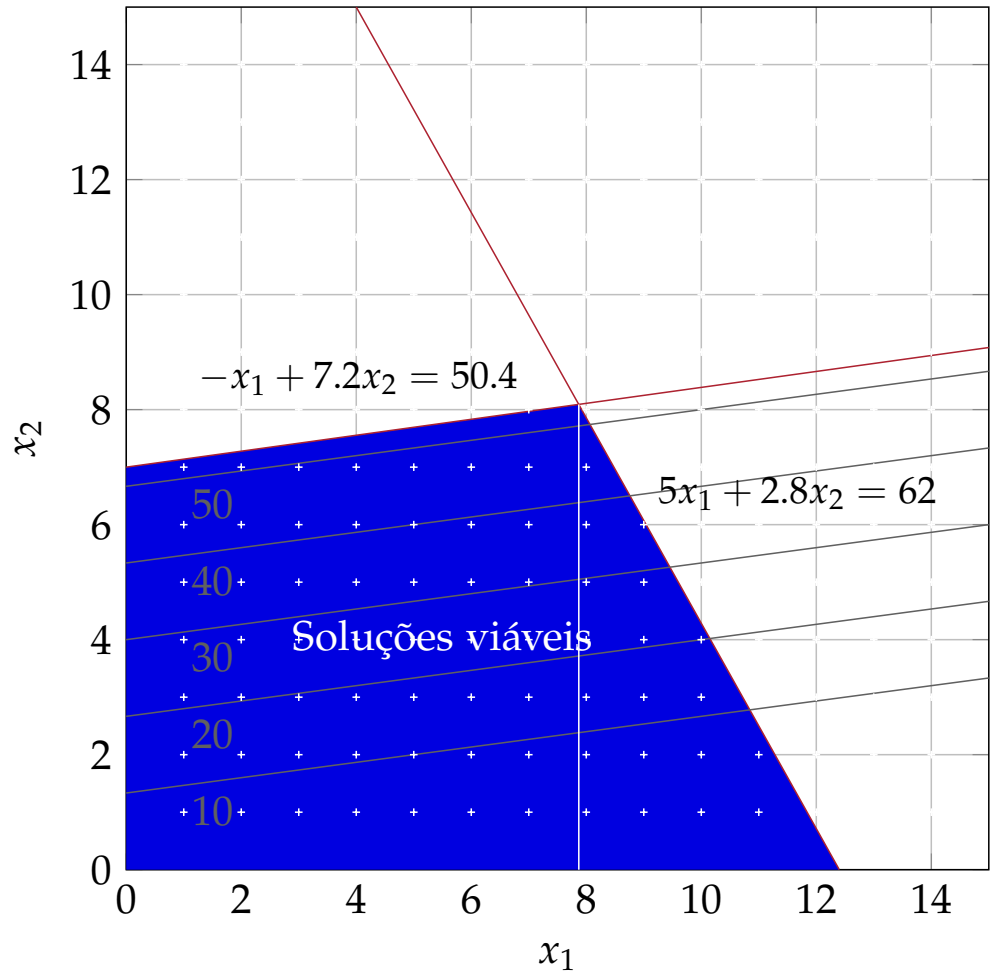
- Solução ótima agora: $x_1 \approx 7.10$, $x_2 \approx 5.17$, $V = 12.28$.

- Será que $\lfloor x_1 \rfloor, \lfloor x_2 \rfloor$ é a solução ótima do PI?

Exemplo

$$\begin{aligned} \textbf{maximiza} \quad & -x_1 + 7.5x_2 \\ \textbf{sujeito a} \quad & -x_1 + 7.2x_2 \leq 50.4, \\ & 5x_1 + 2.8x_2 \leq 62. \end{aligned}$$

Exemplo



- Solução ótima agora: $x_1 \approx 7.87$, $x_2 \approx 8.09$, $V = 52.83$.
- $\lfloor x_1 \rfloor = 7$, $\lfloor x_2 \rfloor = 8$.
- Solução ótima inteira: $x_1 = 0, x_2 = 7$!
- Infelizmente a solução ótima inteira pode ser arbitrariamente distante!

Métodos para resolver PI

- Prove que a solução da relaxação linear sempre é inteira.
- Insere cortes.
- Branch-and-bound.

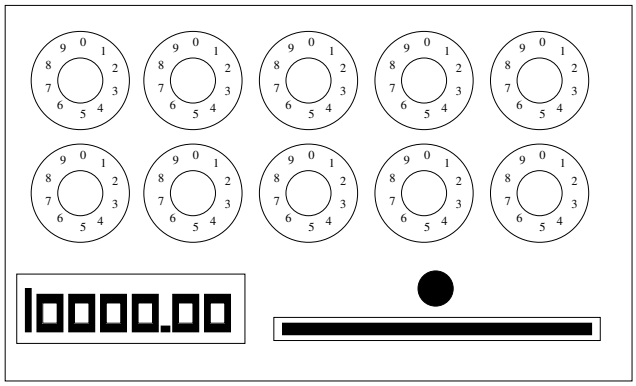
5.2. Motivação e exemplos

Motivação

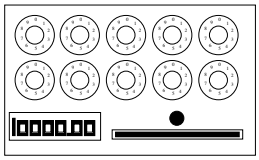
- Otimização combinatória é o ramo da ciência da computação que estuda problemas de otimização em conjuntos (wikipedia).
- "The discipline of applying advanced analytical methods to help make better decisions" (INFORMS)
- Tais problemas são extremamente frequentes e importantes.

Máquina de fazer dinheiro

- Imagine uma máquina com 10 botões, cada botão podendo ser ajustado em um número entre 0 e 9.



Máquina de fazer dinheiro



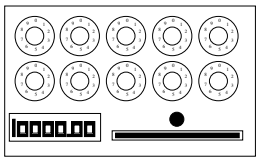
- há uma configuração que retorna R$ 10.000.
- total de combinações: 10^{10}.
- dez testes por segundo
- em um ano:$\Rightarrow 10 \times 60 \times 60 \times 24 \times 365 \cong 3 \times 10^8$

Explosão combinatória

Funções típicas:

n	$\log n$	$n^0.5$	n^2	2^n	$n!$
10	3.32	3.16	10^2	1.02×10^3	3.6×10^6
100	6.64	10.00	10^4	1.27×10^{30}	9.33×10^{157}
1000	9.97	31.62	10^6	1.07×10^{301}	4.02×10^{2567}

"Conclusões"



- Melhor não aceitar a máquina de dinheiro.
- Problemas combinatórios são difíceis.

5.3. Aplicações

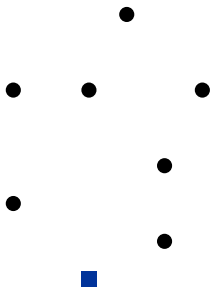
Apanhado de problemas de otimização combinatória

- Caixeiro viajante
- Roteamento

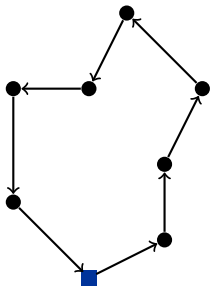
[1] retirado de Integer Programming - Wolsey (1998)

- Projeto de redes
- Alocação de horários
- Tabelas esportivas
- Gestão da produção
- etc.

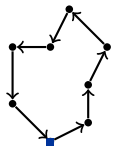
Caixeiro Viajante



Caixeiro Viajante



Caixeiro Viajante



- Humanos são capazes de produzir boas soluções em pouco tempo!
- Humanos ?

Caixeiro Viajante

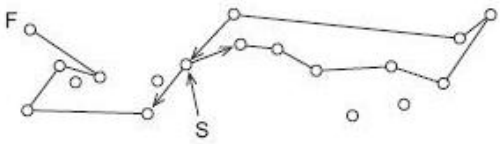


Figure 1.40 Chimpanzee tour (Bido).

Fonte: Applegate et al. (2007)

Caixeiro Viajante

Figure 1.41 Pigeon solving a TSP. Images courtesy of Brett Gibson.

Fonte: Applegate et al. (2007)

Caixeiro Viajante

Fonte: Applegate et al. (2007)

Caixeiro Viajante

> Business leads the traveling salesman here and there, and there is not a good tour for all occurring cases; but through an expedient choice division of the tour so much time can be won that we feel compelled to give guidelines about this. Everyone should use as much of the advice as he thinks useful for his application. We believe we can ensure as much that it will not be possible to plan the tours through Germany in consideration of the distances and the traveling back and fourth, which deserves the traveler's special attention, with more economy. The main thing to remember is always to visit as many localities as possible without having to touch them twice.

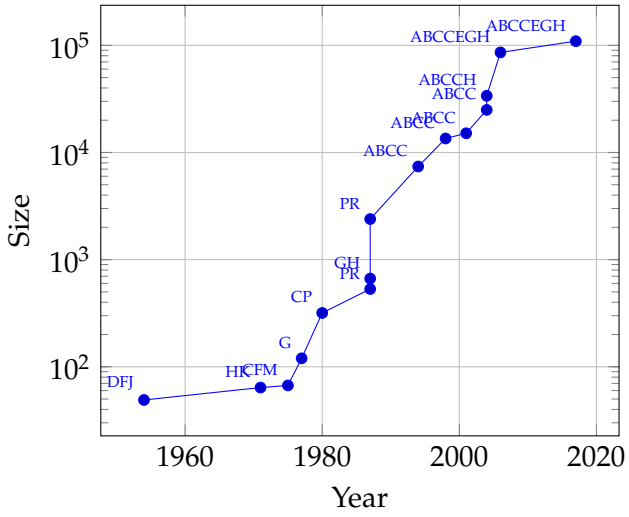

Der
Handlungsreisende
wie er sein soll
und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein.
Von
einem alten Commis - Voyageur.
Mit einem Titelkupfer.
Ilmenau 1832,
Druck und Verlag von B. Fr. Voigt.

Fonte: Applegate et al. (2007)

"Der Handlungsreisende wie er sein soll und was er zu tun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiss zu sein. Von einem alten Commis-Voyageur" (O caixeiro viajante, como ele deve ser e o que ele deve fazer para obter encomendas e garantir um sucesso feliz dos seus negócios. Por um caixeiro viajante experiente).

First brought to the attention of the TSP research community in 1983 by Heiner Muller-Merbach [410]. The title page of this small book is shown in Figure 1.1. The Commis-Voyageur [132] explicitly described the need for good tours in the following passage, translated from the German original by Linda Cook.

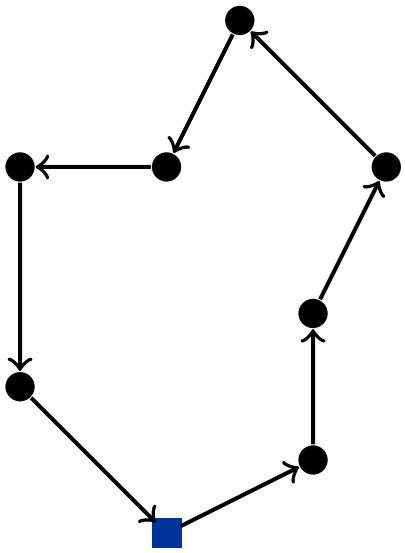
Caixeiro Viajante



Fonte: Applegate et al. (2007)

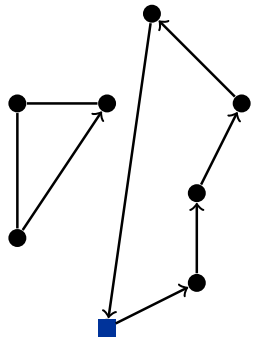
Formulando matemáticamente o PCV

- Associar uma variável a cada possível decisão.



Formulando matemáticamente o PCV

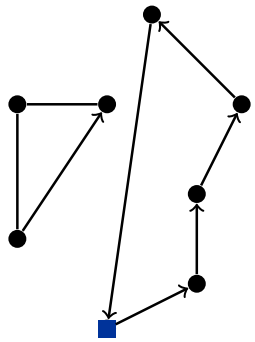
- Associar uma variável a cada possível decisão.



$$\begin{aligned}
\textbf{minimiza} \quad & \sum_{i,j \in N} c_{ij} y_{ij} \\
\textbf{sujeito a} \quad & \sum_{j \in N} x_{ij} + \sum_{j \in N} x_{ji} = 2, \qquad \forall i \in N, \\
& x_{ij} \in \mathbb{B}, \qquad \forall i, j \in N.
\end{aligned}$$

Formulando matemáticamente o PCV

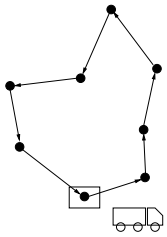
- Associar uma variável a cada possível decisão.

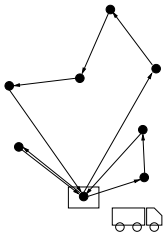


$$\begin{aligned}
\textbf{minimiza} \quad & \sum_{i,j \in N} c_{ij} y_{ij} \\
\textbf{sujeito a} \quad & \sum_{j \in N} x_{ij} + \sum_{j \in N} x_{ji} = 2, \qquad \forall i \in N \\
& x_{ij} \in \mathbb{B}, \qquad \forall i, j \in N.
\end{aligned}$$

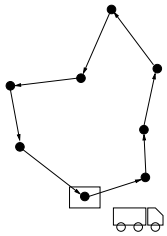
\+ restrições de eliminação de sub-ciclos!

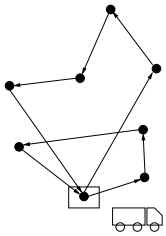
Problemas de roteamento

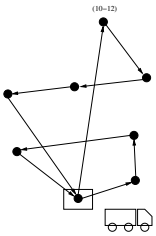


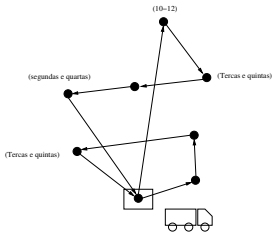


Problemas de roteamento



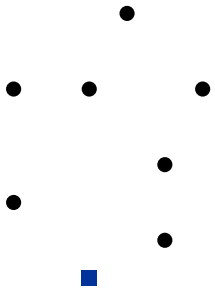




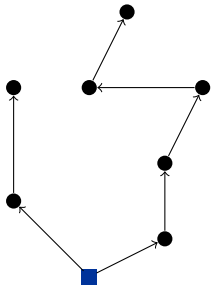


Etc.

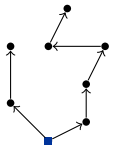
Problemas em árvores



Problemas em árvores



Problemas em árvores - aplicações



- Telecomunicações
- Redes de acesso local
- Engenharias elétrica, civil, etc..

Alocação de tripulações

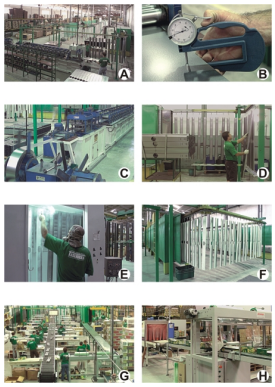
Tabelas esportivas

Proximos Adversários

Fla	Vasco	Paysandu	Criciuma	Vitória
JUVENTUDE	Ponte	Coritiba	**GALO**	**CORINTHIANS**
Guarani	**CRUZEIRO**	**PALMEIRAS**	Santos	Juventude
GALO	São Paulo	Paraná	**FURACÃO**	**GUARANI**
Botafogo	**GOIÁS**	**CRICIÚMA**	Paysandu	Grêmio
PALMEIRAS	Juventude	Santos	**PONTE**	**COXA**
Coritiba	**CORINTHIANS**	**GALO**	Paraná	São Paulo
S. PAULO	Furacão	Guarani	**PALMEIRAS**	**CRUZEIRO**
Cruzeiro	**SANTOS**	**JUVENTUDE**	Coxa	Ponte

Botafogo	Galo	Paraná	Grêmio	Guarani
Cruzeiro	Criciúma	**S.CAETANO**	Palmeiras	Goiás
S. PAULO	**GOIÁS**	Grêmio	**PARANÁ**	**FLA**
Coxa	Fla	**PAYSANDU**	Ponte	Vitória
FLA	**PARANÁ**	Galo	**VITÓRIA**	**PALMEIRAS**
Guarani	**FIGUEIRA**	Goiás	Furacão	**BOTAFOGO**
JUVENTUDE	Paysandu	**CRICIÚMA**	**SANTOS**	Figueira
Corinthians	**GRÊMIO**	Flu	Galo	**PAYSANDU**
FURACÃO	S. Caetano	**INTER**	**GUARANI**	Grêmio

Gestão da produção



Etc.

- programação de projetos
- rotação de plantações
- alocação de facilidades (escolas, centros de comércio, ambulâncias...)
- projeto de circuitos integrados
- portfolio de ações
- etc, etc, etc, etc...

6. Formulação

6.1. Exemplos

"Regras de formulação"

- Criar (boas) formulações é uma arte.
- Algumas diretivas básicas:
  - escolha das variáveis **de decisão**.
  - escolha do objetivo.
  - ajuste das restrições.

Exemplo: 0-1-**Knapsack**

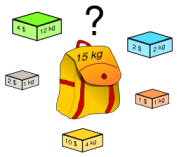
PROBLEMA DA MOCHILA (KNAPSACK)

Instância Um conjunto de n itens com valores v_i e pesos p_i, $i \in [n]$. Um limite de peso P do mochila.

Solução Um conjunto $S \subseteq [n]$ de itens que cabe na mochila, i.e. $\sum_{i \in S} p_i \leq P$.

Objetivo Maximizar o valor $\sum_{i \in S} v_i$.

- Observação: Existe uma solução (pseudo-polinomial) com programação dinâmica em tempo $O(Pn)$ usando espaço $O(P)$.



Formulação – Problema da mochila

$$\begin{aligned} \textbf{maximiza} \quad & \sum_{i \in [n]} v_i x_i \\ \textbf{sujeito a} \quad & \sum_{i \in [n]} p_i x_i \leq P, \\ & x_i \in \mathbb{B}. \end{aligned}$$

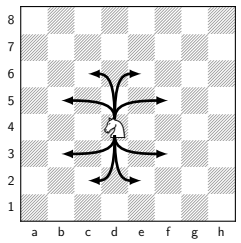


Figura 6.1.: Os campos atacados por um cavalo num tabuleiro de xadrez.

Exemplo 6.1 (Maximizar cavalos num tabuleiro de xadrez)
Qual o número máximo de cavalos que cabe num tabuleiro de xadrez, tal que nenhum ameaça um outro?
Formulação do problema dos cavalos com variáveis indicadores x_{ij}:

$$\begin{aligned}
\textbf{maximiza} \quad & \sum_{i,j\in[8]} x_{ij} \\
\textbf{sujeito a} \quad & x_{ij} + x_{i-2,j+1} \leq 1, && 3 \leq i \leq 8, j \in [7], \\
& x_{ij} + x_{i-1,j+2} \leq 1, && 2 \leq i \leq 8, j \in [6], \\
& x_{ij} + x_{i+2,j+1} \leq 1, && i \in [6], j \in [7], \\
& x_{ij} + x_{i+1,j+2} \leq 1, && i \in [7], j \in [6].
\end{aligned}$$

Número de soluções do problema dos cavalos (A030978)

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
k	1	4	5	8	13	18	25	32	41	50	61	72	85	98	113

◊

6.2. Técnicas para formular programas inteiros

Um problema recorrente com indicadores $x_1, \ldots, x_n \in \mathbb{B}$ e selecionar no máximo, exatamente, ou no mínimo k dos n itens. As restrições

$$\sum_{i\in[n]} x_i \leq k; \qquad \sum_{i\in[n]} x_i = k; \qquad \sum_{i\in[n]} x_i \geq k$$

conseguem isso.

Exemplo 6.2 (Localização de facilidades simples 1)
Em n cidades dadas queremos instalar no máximo k fábricas ($k \leq n$) de modo a minimizar o custo da instalação das fábricas. A instalação na cidade $j \in [n]$ custa f_j. Podemos usar indicadores para $y_j \in \mathbb{B}$ para a instalação da uma fábrica na cidade j e formular

$$\begin{aligned}
\textbf{minimiza} \quad & \sum_{j\in[n]} f_j y_j \\
\textbf{sujeito a} \quad & \sum_{j\in[n]} y_j = k, \\
& y_j \in \mathbb{B}, && j \in [n].
\end{aligned}$$

(Obviamente para resolver este problema é suficiente escolher as k cidades de menor custo. No exemplo 6.3 estenderemos esta formulação para incluir custos de transporte.) ◊

6.2.1. Formular restrições lógicas

Formulação: Indicadores

- Variáveis indicadores $x, y \in \mathbb{B}$: Seleção de um objeto.
- Implicação (limitada): Se x for selecionado, então y deve ser selecionado

$$x \leq y, \qquad x, y \in \mathbb{B}$$

- Ou (disjunção):

$$x + y \geq 1, \qquad x, y \in \mathbb{B}$$

- Ou-exlusivo:

$$x + y = 1, \qquad x, y \in \mathbb{B}$$

Exemplo 6.3 (Localização de facilidades não-capacitado)
Queremos incluir no exemplo 6.2 clientes. Suponha que em cada cidade tem um cliente, e queremos, junto com os custos das fábricas instaladas, minimizar o custo de atendimento dos clientes. Entre cada par de cidade, i e j, o custo de transporte é dado por c_{ij} (ver Figura 6.2). Para formulação escolhemos variáveis de decisão $x_{ij} \in \mathbb{B}$, que indicam se o cliente i for atendido pela fábrica em j. É importante "vincular" as variáveis de decisão: o cliente i pode ser atendido pela cidade j somente se na cidade j foi instalada uma fábrica, i.e. $x_{ij} \to y_j$.

(a) Exemplo de uma instância (b) Exemplo de uma solução

Figura 6.2.: Localização de facilidades.

$$
\begin{array}{lll}
\textbf{minimiza} & \displaystyle\sum_{j\in[n]} f_j y_j + \sum_{i,j\in[n]} c_{ij} x_{ij} & \\
\textbf{sujeito a} & \displaystyle\sum_{j\in[n]} x_{ij} = 1, \qquad i \in [n], & \text{(só uma fábrica atende)} \\
& \displaystyle\sum_{j\in[n]} y_j \leq m, & \text{(no máximo } m \text{ fábricas)} \\
& x_{ij} \leq y_j, \qquad i \in [n], j \in [n], & \text{(só fáb. existentes atendem)} \\
& x_{ij} \in \mathbb{B}, \qquad i \in [n], j \in [n], & \\
& y_j \in \mathbb{B}, \qquad j \in [n]. &
\end{array}
$$

$\Diamond$

Formulação: Indicadores
Para $x, y, z \in \mathbb{B}$

- Conjunção $x = yz = y \wedge z$

$$
\begin{aligned}
x &\leq (y+z)/2 \\
x &\geq y + z - 1
\end{aligned}
\tag{6.1}
$$

- Disjunção $x = y \vee z$

$$
\begin{aligned}
x &\geq (y+z)/2 \\
x &\leq y + z
\end{aligned}
\tag{6.2}
$$

- Negação $x = \neg y$

$$
x = 1 - y \tag{6.3}
$$

- Implicação: $z = x \to y$

$$z \leq 1 - x + y \tag{6.4}$$
$$z \geq (1 - x + y)/2 \tag{6.5}$$

Exemplo 6.4 (Max-3-SAT)
Seja $\varphi(x_1, \ldots, x_n) = \bigwedge_{i\in[m]} c_i$ uma fórmula em forma normal conjuntiva, com cláusulas da forma $c_i = l_{i1} \vee l_{i2} \vee l_{i3}$. Queremos encontrar uma atribuição $x_i \in \{f, v\}$ maximizando o número de cláusulas satisfeitas.
Seja $\bar{c}_i \in \mathbb{B}$ uma variável que indica que cláusula c_i é satisfeita. Também vamos introduzir uma variável binária $\bar{x}_k \in \mathbb{B}$ para cada variável x_k do problema, e uma variável auxiliar $\bar{l}_{ij}$ para literal l_{ij} do problema.

$$\begin{aligned}
\textbf{maximiza} \quad & \sum_{i\in[m]} \bar{c}_i \\
\textbf{sujeito a} \quad & \bar{c}_i \leq \bar{l}_{i1} + \bar{l}_{i2} + \bar{l}_{i3}, \\
& \bar{l}_{ij} = \bar{x}_k, && \text{caso } l_{ij} = x_k, \\
& \bar{l}_{ij} = 1 - \bar{x}_k, && \text{caso } l_{ij} = \neg x_k, \\
& \bar{c}_i \in \mathbb{B}, \bar{x}_k \in \mathbb{B}, \bar{l}_{ij} \in \mathbb{B}.
\end{aligned}$$

◊

6.2.2. Formular restrições condicionais

Indicadores para igualdades satisfeitas Queremos definir uma variável $y \in \mathbb{B}$ que indica se uma dada restrição é satisfeita.

- Para $\sum_{i\in[n]} a_i x_i \leq b$: Escolhe um limite superior M para $\sum_{i\in[n]} a_i x_i - b$, um limite inferior m para $\sum_{i\in[n]} a_i x_i - b$ e uma constante $\epsilon > 0$ pequena.

$$\begin{aligned}
\sum_{i\in[n]} a_i x_i &\leq b + M(1 - y) \\
\sum_{i\in[n]} a_i x_i &\geq b + my + (1 - y)\epsilon
\end{aligned} \tag{6.6}$$

- Para $x > 0$: Escolhe um limite superior M para x e uma constante ϵ pequena.

$$\begin{aligned}
x &\geq \epsilon y, \\
x &\leq My.
\end{aligned} \tag{6.7}$$

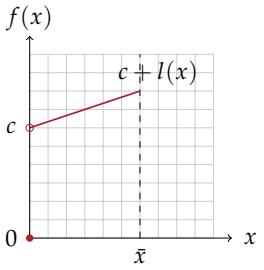


Figura 6.3.: Função objetivo não-linear

Exemplo 6.5 (Custos fixos)
Uma aplicação para problemas de minimização com uma função objetivo não-linear. Queremos minimizar custos, com uma "entrada" fixa c da forma

$$f(x) = \begin{cases} 0 & \text{caso } x = 0 \\ c + l(x) & \text{caso } 0 < x \leq \bar{x} \end{cases}$$

e $l(x)$ uma função linear (ver Figura 6.3). Com uma $y \in \mathbb{B}$ indica a positividade de x, i.e. $y = 1$ sse $x > 0$ podemos definir a função objetivo por

$$f(x) = cy + l(x)$$

e a técnica da equação (6.7) resolve o problema. Como o objetivo é minimizar $f(x)$ a primeira equação $x \geq \epsilon y$ é redundante: caso $y = 1$ não faz sentido escolher uma solução com $x = 0$, porque para $x = 0$ existe a solução de menor custo $x = y = 0$. Logo

$$x \leq \bar{x}y,$$
$$x \in \mathbb{R}, y \in \mathbb{B},$$

é suficiente neste caso.

◊

Exemplo
Planejamento de produção (ingl. uncapacitated lot sizing)

- Objetivo: Planejar a futura produção no próximos n semanas.
- Parâmetros: Para cada semana $i \in [n]$
  - Custo fixo f_i para produzir,
  - Custo p_i para produzir uma unidade,
  - Custo h_i por unidade para armazenar,
  - Demanda d_i

Exemplo
Seja

- x_i a quantidade produzida,

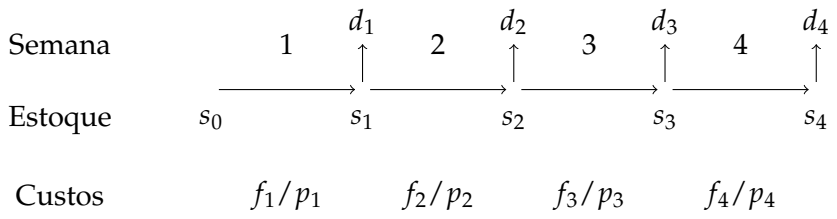


Figura 6.4.: Planejamento de produção.

- s_i a quantidade no estoque no final da semana i,
- $y_i = 1$ sem tem produção na semana i, 0 senão.

Problema:

- Função objetivo tem custos fixos, mas x_i não tem limite.
- Determina ou estima um valor limite M.

Exemplo

$$
\begin{aligned}
\textbf{minimiza} \quad & \sum_{i\in[n]} p_i x_i + h_i s_i + f_i y_i \\
\textbf{sujeito a} \quad & s_i = s_{i-1} + x_i - d_i, && i \in [n], \\
& s_0 = 0, \\
& x_i \leq M y_i, && i \in [n], \\
& x \in \mathbb{R}^n, y \in \mathbb{B}^n.
\end{aligned}
$$

Disjunção de equações

- Queremos que aplica-se uma das equações

$$
\begin{aligned}
f_1 &\leq f_2, \\
g_1 &\leq g_2.
\end{aligned}
$$

- Solução, com constante M suficientemente grande

$$\begin{aligned} &f_1 \leq f_2 + Mx, \\ &g_1 \leq g_2 + M(1-x), \\ &x \in \mathbb{B}. \end{aligned}$$

6.3. Formulações alternativas

Uma problema de programação linear ou inteira geralmente possui mais que uma formulação. A Figura 6.5 mostra diversas formulações que definem o mesmo conjunto de soluções inteiras.

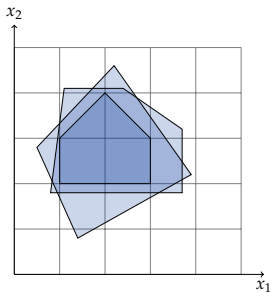


Figura 6.5.: Diferentes formulações lineares que definem o mesmo conjunto de soluções inteiras.

Na programação linear existe pouca diferença entre as formulações: a solução é a mesma e o tempo para resolver o problema é comparável, para um número comparável de restrições e variáveis. Na programação inteira uma formulação boa é mais importante. Como a solução de programas inteiras é NP-completo, frequentemente a relaxação linear é usada para obter uma aproximação. Diferentes formulação de um programa inteiro possuem diferentes qualidades da relaxação linear. Uma maneira de quantificar a qualidade de uma formulação é o *gap de integralidade*(ingl. integrality gap). Para um problema P e uma instância $i \in P$ seja $\mathrm{OPT}(i)$ a solução ótima inteira e $LP(i)$ a solução da relaxação linear. O gap de integralidade é

$$g(P) = \sup_{i \in P} \frac{\mathrm{LP}(i)}{\mathrm{OPT}(i)} \tag{6.8}$$

(para um problema de maximização.) O gap de integralidade dá uma garantia para qualidade da solução da relaxação linear: caso o gap é g, a solução não é mais que um fator g maior que a solução integral ótima.

Exemplo 6.6 (Conjunto independente máximo)
Uma formulação do problema de encontrar o conjunto independente máximo num grafo não-direcionado $G = (V, A)$ é

$$\begin{aligned} \textbf{maximiza} \quad & \sum_{v \in V} x_v, && && \text{(CIM)} \\ \textbf{sujeito a} \quad & x_u + x_v \leq 1, && \forall \{u, v\} \in E, \\ & x_v \in \mathbb{B}, && \forall v \in V. \end{aligned}$$

No grafo completo com n vértices K_n a relaxação linear possui um valor pelo menos $n/2$ (porque a solução $x_v = 1/2, v \in V$ possui valor $n/2$), enquanto

a solução ótima inteira é 1. Por isso, o programa (CIM) possui um gap de integralidade ilimitado. ◊

6.4. Exercícios

(Soluções a partir da página 222.)

Exercício 6.1
A empresa "Festa fulminante" organiza festas. Nos próximos n dias, ela precisa p_i pratos, $1 \leq i \leq n$. No começo de cada dia gerente tem os seguintes opções:

- Comprar um prato para um preço de c reais.
- Mandar lavar um prato devagarmente em d_1 dias, por um preço de l_1 reais.
- Mandar lavar um prato rapidamente em $d_2 < d_1$ dias, por um preço de $l_2 > l_1$ reais.

O gerente quer minimizar os custos dos pratos. Formule como programa inteira.

Exercício 6.2
Para os problemas abaixo, encontra uma formulação como programa inteira.

Conjunto independente máximo

Instância Um grafo não-direcionado $G = (V, A)$.

Solução Um *conjunto independente* I, i.e. $I \subseteq V$ tal que para vértices $v_1, v_2 \in I$, $\{v_1, v_2\} \notin A$.

Objetivo Maximiza $|I|$.

Emparelhamento perfeito com peso máximo

Instância Um grafo não-direcionado bi-partido $G = (V_1 \dot\cup V_2, A)$ (a fato de ser bi-partido significa que $A \subseteq V_1 \times V_2$) com pesos $p : A \to \mathbb{R}$ nos arcos.

Solução Um *emparelhamento perfeito*, i.e. um conjunto de arcos $C \subseteq A$ tal

que todos nós no sub-grafo $G[C] = (V_1 \cup V_2, C)$ tem grau 1.

Objetivo Maximiza o peso total $\sum_{c \in C} p(c)$ do emparelhamento.

PROBLEMA DE TRANSPORTE

Instância n depósitos, cada um com um estoque de p_i produtos, $i \in [n]$, e m clientes, cada um com uma demanda d_j, $j \in [m]$ produtos. Custos de transporte a_{ij} de cada depósito $i \in [n]$ para cada cliente $j \in [m]$.

Solução Um decisão quantos produtos x_{ij} devem ser transportados do depósito $i \in [n]$ ao cliente $j \in [m]$, que satisfaz (i) Cada depósito manda todo seu estoque (ii) Cada cliente recebe exatamente a sua demanda. (Observe que o número de produtos transportados deve ser integral.)

Objetivo Minimizar os custos de transporte $\sum_{i \in [n], j \in [m]} a_{ij} x_{ij}$.

CONJUNTO DOMINANTE

Instância Um grafo não-direcionado $G = (V, A)$.

Solução Um *conjunto dominante*, i.e. um conjunto $D \subseteq V$, tal que $\forall v \in V : v \in D \vee (\exists u \in D : \{u, v\} \in A)$ (cada vértice faz parte do conjunto dominante ou tem um vizinho no conjunto dominante).

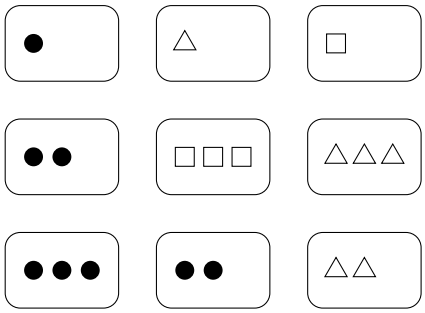
Objetivo Minimizar o tamanho do conjunto dominante $|D|$.

Exercício 6.3
Acha uma formulação inteira para todos os 21 problemas que o Karp provou NP-completo (Karp. 1972).

Exercício 6.4
Juliano é fã do programa de auditório Apagando e Ganhando, um programa no qual os participantes são selecionados atráves de um sorteio e recebem prêmios em dinheiro por participarem. No programa, o apresentador escreve um número de N dígitos em uma lousa. O participante então deve apagar exatamente D dígitos do número que está na lousa; o número formado pelos

dígitos que restaram é então o prêmio do participante. Juliano finalmente foi selecionado para participar do programa, e pediu que você escrevesse um programa inteira que, dados o número que o apresentador escreveu na lousa, e quantos dígitos Juliano tem que apagar, determina o valor do maior prêmio que Juliano pode ganhar.
(Fonte: Maratona de programação regional 2008, RS)

Exercício 6.5
Set é um jogo jogado com um baralho no qual cada carta pode ter uma, duas ou três figuras. Todas as figuras em uma carta são iguais, e podem ser círculos, quadrados ou triângulos. Um set é um conjunto de três cartas em que, para cada característica (número e figura), u ou as três cartas são iguais, ou as três cartas são diferentes. Por exemplo, na figura abaixo, (a) é um set válido, já que todas as cartas têm o mesmo tipo de figura e todas elas têm números diferentes de figuras. Em (b), tanto as figuras quanto os números são diferentes para cada carta. Por outro lado, (c) não é um set, já que as duas ultimas cartas têm a mesma figura, mas esta é diferente da figura da primeira carta.



(a) (b) (c)

O objetivo do jogo é formar o maior número de sets com as cartas que estão na mesa; cada vez que um set é formado, as três cartas correspondentes são removidas de jogo. Quando há poucas cartas na mesa, é fácil determinar o maior número de sets que podem ser formados; no entanto, quando há muitas cartas há muitas combinações possíveis. Seu colega quer treinar para o campeonato mundial de Set, e por isso pediu que você fizesse um programa inteira e que calcula o maior número de sets que podem ser formados com um determinado conjunto de cartas.
(Fonte: Maratona de programação regional 2008, RS)

Exercício 6.6
Para os problemas abaixo, acha uma formulação como programa inteiro.

COBERTURA POR ARCOS

Instância Um grafo não-direcionado $G = (V, E)$ com pesos $c : E \to \mathbb{Q}$ nos arcos.

Solução Uma cobertura por arcos, i.e. um subconjunto $E' \subseteq E$ dos arcos tal que todo vértice faz parte de ao menos um arco selecionado.

Objetivo Minimiza o custo total dos arcos selecionados em E'.

CONJUNTO DOMINANTE DE ARCOS

Instância Um grafo não-direcionado $G = (V, E)$ com pesos $c : E \to \mathbb{Q}$ nos arcos.

Solução Um conjunto dominante de arcos, i.e. um subconjunto $E' \subseteq E$ dos arcos tal que todo arco compartilha um vértico com ao menos um arco em E'.

Objetivo Minimiza o custo total dos arcos selecionados em E'.

COLORAÇÃO DE GRAFOS

Instância Um grafo não-direcionado $G = (V, E)$.

Solução Uma coloração do grafo, i.e. uma atribuição de cores nas vértices $c : V \to \mathbb{Z}$ tal que cada par de vértices ligando por um arco recebe uma cor diferente.

Objetivo Minimiza o número de cores diferentes.

CLIQUE MÍNIMO PONDERADO

Instância Um grafo não-direcionado $G = (V, E)$ com pesos $c : V \to \mathbb{Q}$ nos vértices.

Solução Uma *clique*, i.e. um subconjunto $V' \subseteq V$ de vértices tal que

existe um arco entre todo par de vértices em V'.

Objetivo Minimiza o peso total dos vértices selecionados V'.

SUBGRAFO CÚBICO

Instância Um grafo não-direcionado $G = (V, E)$.

Solução Uma subgrafo cúbico, i.e. uma seleção $E' \subseteq E$ dos arcos, tal que cada vértice em $G' = (V, E')$ possui grau 0 ou 3.

Objetivo Minimiza o número de arcos selecionados $|E'|$.

Exercício 6.7
Uma empresa tem que decidir quais de sete investimentos devem ser feitos. Cada investimento pode ser feito somente uma única vez. Os investimentos tem lucros (ao longo prazo) e custos iniciais diferentes como segue.

	Investimento						
	1	2	3	4	5	6	7
Lucro estimado [MR$]	17	10	15	19	7	13	9
Custos iniciais [MR$]	43	28	34	48	17	32	23

A empresa tem 100 MR$ capital disponível. Como maximizar o lucro total (ao longo prazo, não considerando os investimentos atuais), respeitando que os investimentos 1, 2 e 3, 4 são mutualmente exclusivas, e nem o investimento 3 nem o investimento 4 pode ser feita, sem ao menos um investimento em 1 ou 2 (as outros investimentos não tem restrições).

Exercício 6.8
Um produtor de brinquedos projetou dois novos brinquedos para Natal. A preparação de uma fábrica para produzir custaria R$ 50000 para a primeiro brinquedo e R$ 80000 para o segundo. Após esse investimento inicial, o primeiro brinquedo rende R$ 10 por unidade e o segundo R$ 15.
O produtor tem duas fábricas disponíveis mas pretende usar somente uma, para evitar custos de preparação duplos. Se a decisão for tomada de produzir os dois brinquedos, a mesma fábrica seria usada.

Por hora, a fábrica 1 é capaz de produzir 50 unidades do brinquedo 1 ou 40 unidades do brinquedo 2 e tem 500 horas de produção disponível antes de Natal. A fábrica 2 é capaz de produzir 40 unidades do brinquedo 1 ou 25 unidades do brinquedo 2 por hora, e tem 700 horas de produção disponível antes de Natal.
Como não sabemos se os brinquedos serão continuados depois Natal, a problema é determinar quantas unidades de cada brinquedo devem ser produzidas até Natal (incluindo o caso de um brinquedo não sendo produzido) de forma que maximiza o lucro total.

Exercício 6.9
Uma empresa produz pequenos aviões para gerentes. Os gerentes frequentemente precisam um avião com características específicas que gera custos inicias altos no começo da produção. A empresa recebeu encomendas para três tipos de aviões de três clientes, mas como ela está com capacidade de produção limitada, ela tem que decidir quais das três aviões ela vai produzir. Os seguintes dados são relevantes

Aviões	Cliente		
produzidas	1	2	3
Custo inicial [MR$]	3	2	0
Lucro [MR$/avião]	2	3	0.8
Capacidade usada [%/avião]	20	40	20
Demanda máxima [aviões]	3	2	5

Os clientes aceitam qualquer número de aviões até a demanda máxima. A empresa tem quer decidir quais e quantas aviões ela vai produzir. As aviões serão produzidos em paralelo.

Exercício 6.10 (Winkler)
Uma fechadura de combinação com três discos, cada um com números entre 1 e 8, possui um defeito, tal que precisa-se somente dois números corretos dos três para abri-la. Qual o número mínimo de combinações (de três números) que precisa-se testar, para garantidamente abrir a fechadura?
Formule um programa inteiro e resolva-o.

Exercício 6.11
Formule o problema

MAX-k-SAT

Entrada Uma fórmula em forma normal conjuntiva com m variáveis e n cláusulas $\varphi(x_1, \ldots, x_m) = C_1 \wedge \cdots \wedge C_n$ tal que cada cláusula possui no máximo k literais

Solução Uma atribuição $x_i \mapsto \mathbb{B}$.

Objetivo Maximizar o número de cláusulas satisfeitas.

(Dica: Usa as desigualdades (6.1)–(6.3). Começa com $k = 3$.)

Exercício 6.12
A Seção 6.2.1 mostrava como expressar a restrição lógica $z = x \wedge y$ linearmente. A formulação linear precisava duas restrições lineares. Mostra que não existe uma única restrição linear que é suficiente para expressar $z = x \wedge y$.
(Dica: Supõe que $z = ax + by + c$ (ou $z \geq ax + by + c$, ou $z \leq ax + by + c$) com constantes a, b, c e mostra que as restrições que resultam de uma análise caso a caso levam a uma contradição ou não são suficientes para garantir a restrição lógica.)

Exercício 6.13
Considere o problema de coloração de grafos:

Coloração de grafos

Instância Um grafo não-direcionado $G = (V, E)$.

Solução Uma coloração do grafo, i.e. uma atribuição de cores às vértices $c : V \to \mathbb{Z}_+$ tal que cada par de vértices ligado por uma aresta recebe uma cor diferente.

Objetivo Minimiza o número de cores diferentes.

Uma formulação possível é introduzir uma variável $x_{vc} \in \mathbb{B}$ tal que $x_{vc} = 1$ caso o vértice $v \in V$ recebe a cor c. Como nunca tem mais que $n = |V|$ cores, podemos escolher $C = [n]$. Temos a condição

$$\sum_{c \in C} x_{vc} = 1, \qquad \forall v \in V. \tag{6.9}$$

Uma coloração válida ainda tem que satisfazer

$$x_{uc} + x_{vc} \leq 1, \qquad \forall \{u, v\} \in E, c \in C. \tag{6.10}$$

Para contar o número de cores vamos usar variáveis auxiliares $u_c \in \mathbb{B}$ com $u_c = 1$ caso a cor $c \in C$ foi usada. Eles satisfazem

$$u_c \geq \sum_{v \in V} x_{vc}/n, \qquad \forall c \in C. \tag{6.11}$$

Com isso obtemos

$$\begin{aligned} (C_1)\ \textbf{minimiza} \quad & \sum_{c \in C} u_c, \\ \textbf{sujeito a} \quad & (6.9), (6.10), (6.11) \\ & x_{vc} \in \mathbb{B}, u_c \in \mathbb{B}, \qquad \forall v \in V, c \in C. \end{aligned}$$

Um outro modelo é minimizar a soma das cores. Seja $f_v \in \mathbb{Z}_+$ a cor do vértice $v \in V$, que pode ser definida por

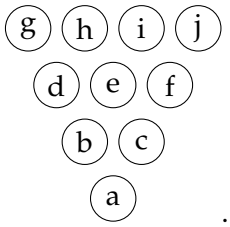
$$f_v = \sum_{c \in C} c x_{vc}, \qquad \forall v \in V. \tag{6.12}$$

Com isso podemos formular

$$\begin{aligned} (C_2)\ \textbf{minimiza} \quad & \sum_{v \in V} f_v, \\ \textbf{sujeito a} \quad & (6.9), (6.10), (6.12), \\ & x_{vc} \in \mathbb{B}, f_c \in \mathbb{Z}_+, \qquad \forall v \in V, c \in C. \end{aligned}$$

Os modelos (C_1) e (C_2) são equivalentes?

Exercício 6.14
Considere o problema de posicionar os números $1, \ldots, 10$ nas posições $P = \{a, \ldots, j\}$ do triângulo



.

Um colega afirma que podemos usar variáveis $x_a, \ldots, x_j \in \mathbb{Z}$ e as restrições

$$\begin{aligned} & 1 \leq x_p \leq 10, && \forall p \in P, \\ & \sum_{p \in P} x_p = 55, \\ & \prod_{p \in P} x_p = 10! \end{aligned}$$

Ele tem razão?

Exercício 6.15
Aplica as técnicas da Seção 6.2.1 para derivar uma formulação do problema MAX-3-SAT discutido no Exemplo 6.4. Compara as duas formulações.

7. Técnicas de solução

7.1. Introdução

Limites

- Exemplo: Problema de maximização.
- Limite inferior (limite primal): Cada solução viável.
  - Qualquer técnica construtiva, p.ex. algoritmos gulosos, heurísticas etc.
- Limite superior (limite dual): Essencialmente usando uma relaxação
  - Menos restrições $\Rightarrow$ conjunto maior de solução viáveis.
  - Nova função objetivo que é maior ou igual.
- Importante: Relaxação linear: $x \in \mathbb{Z} \Rightarrow x \in \mathbb{R}$.

7.2. Problemas com solução eficiente

Observação 7.1 (Regra de Laplace)
Lembrança: A determinante de uma matriz pela regra de Laplace é

$$\det(A) = \sum_{i \in [n]} (-1)^{i+j} a_{ij} \det(A_{ij}) = \sum_{j \in [n]} (-1)^{i+j} a_{ij} \det(A_{ij})$$

com $j \in [n]$ arbitrário para a primeira variante, e $i \in [n]$ arbitrário para a segunda, e com A_{ij} a submatriz sem linha i e coluna j. $\Diamond$

Relaxação linear

- Solução simples: A relaxação linear possui solução ótima inteira.
- Como garantir?
- Com base B temos a solução $x = (x_B \; x_N)^t = (B^{-1}b, 0)^t$.
- Observação: Se $b \in \mathbb{Z}^m$ e $|\det(B)| = 1$ para a base ótima, então o PL resolve o PI.

Relaxação inteira

- Para ver isso: Regra de Cramer.
- A solução de $Ax = b$ é

$$x_i = \frac{\det(A_i)}{\det(A)}$$

com A_i a matriz resultante da substituição da i-ésima coluna de A por b.

Prova. Seja U_i a matriz identidade com a i-ésima coluna substituído por x, i.e.

$$\begin{pmatrix} 1 & & x_1 & & \\ & \ddots & x_2 & & \\ & & \vdots & & \\ & & x_{n-1} & \ddots & \\ & & x_n & & 1 \end{pmatrix}$$

Temos que $AU_i = A_i$ e com $\det(U_i) = x_i$ temos

$$\det(A_i) = \det(AU_i) = \det(A)\det(U_i) = \det(A)x_i.$$

■

Exemplo: Regra de Cramer

$$\begin{pmatrix} 3 & 2 & 1 \\ 5 & 0 & 2 \\ 2 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Exemplo: Regra de Cramer

$$\begin{vmatrix} 3 & 2 & 1 \\ 5 & 0 & 2 \\ 2 & 1 & 2 \end{vmatrix} = -13; \qquad \begin{vmatrix} 1 & 2 & 1 \\ 1 & 0 & 2 \\ 1 & 1 & 2 \end{vmatrix} = -1$$

$$\begin{vmatrix} 3 & 1 & 1 \\ 5 & 1 & 2 \\ 2 & 1 & 2 \end{vmatrix} = -3; \qquad \begin{vmatrix} 3 & 2 & 1 \\ 5 & 0 & 1 \\ 2 & 1 & 1 \end{vmatrix} = -4$$

Logo $x_1 = 1/13; \quad x_2 = 3/13; \quad x_3 = 4/13.$

Aplicação da regra de Cramer

- Como garantir que $x = B^{-1}b$ é inteiro?
- Cramer:
$$x_i = \frac{\det(B_i)}{\det(B)}$$
- Condição possível: (a) $\det(B_i)$ inteiro, (b) $\det(B) \in \{-1, 1\}$.
- Garantir (a): $A \in \mathbb{Z}^{m \times n}$ e $b \in \mathbb{Z}^m$.
- Garantir (b): Toda submatriz quadrada não-singular de A tem determinante $\{-1, 1\}$.

Exemplo 7.1
Observe que essas condições são suficientes, mas não necessárias. É possível que $Bx = b$ possui solução inteira sem essas condições ser satisfeitas. Por exemplo

$$\begin{pmatrix} 2 & 2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

tem a solução inteira $(x_1\ x_2) = (1\ 0)$, mesmo que $\det(A) = -2$. ◊

A relaxação é inteira

Definição 7.1
Uma matriz quadrada inteira $A \in \mathbb{R}^{n \times n}$ é *unimodular* se $|\det(A)| = 1$. Uma matriz arbitrária A é *totalmente unimodular* (TU) se cada submatriz quadrada não-singular A' de A é modular, i.e. $\det(A') \in \{0, 1, -1\}$.

Uma consequência imediata dessa definição: $a_{ij} \in \{-1, 0, 1\}$.

Exemplo

Quais matrizes são totalmente unimodulares?

$$\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}; \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & -1 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{pmatrix}; \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Exemplo

$$A = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$$

TU? Não: $\det(A) = 2$.

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

TU? Não: $\det(A) = 2$.

7.2.1. Critérios para soluções inteiras

Critérios

Proposição 7.1
Se A é TU então

(i) A^t é TU.

(ii) $(A\ I)$ com matriz de identidade I é TU.

(iii) Uma matriz B que é uma permutação das linhas ou colunas de A é TU.

(iv) Multiplicando uma linha ou coluna por -1 produz uma matriz TU.

Prova. (i) Qualquer submatriz quadrada B^t de A^t e uma submatriz B de A também. Com $\det(B) = \det(B^t)$, segue que A^t é totalmente unimodular. (ii) Qualquer submatriz de (AI) tem a forma $(A'I')$ com A' submatriz de A e I' submatriz de I. Com $|\det(A'I')| = |\det(A')|$ segue que (AI) é TU. (iii) Cada submatriz de B é uma submatriz de A. (iv) A determinante troca no máximo o sinal. ■

Exercício 7.1 pede generalizar a proposição 7.1.

Critérios

Proposição 7.2 (Critério de partição de linhas)
Uma matriz A é totalmente unimodular caso

(i) $a_{ij} \in \{+1, -1, 0\}$

(ii) Cada coluna contém no máximo dois coeficientes não-nulos.

(iii) Existe uma partição de linhas $M_1 \dot\cup M_2 = [1, m]$ tal que cada coluna *com dois coeficientes não-nulos* satisfaz

$$\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} = 0$$

Prova. (da proposição 7.2). Prova por contradição. Seja A uma matriz que satisfaz os critérios da proposição 7.2, e B a menor submatriz quadrada de A tal que $\det(B) \notin \{0, +1, -1\}$. B não contém uma coluna com um único coeficiente não-nulo: seria uma contradição com a minimalidade do B (removendo a linha e a coluna que contém esse coeficiente, obtemos uma matriz

quadrada menor B^*, que ainda satisfaz $\det(B^*) \notin \{0, +1, -1\}$). Logo, B contém dois coeficientes não-nulos em cada coluna. Aplicando a condição (3) acima, subtraindo as linhas com índice em M_1 das linhas com índice em M_2 podemos ver as linhas do B são linearmente dependentes e portanto temos $\det(B) = 0$, uma contradição. ■

Observação 7.2

O critério de partição da linhas é suficiente, mas não necessário. A matriz

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix},$$

por exemplo, é totalmente unimodular, mas o critério não se aplica. ◊

Exemplo 7.2

A matriz

$$\begin{pmatrix} 1 & -1 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{pmatrix}$$

claramente satisfaz os critérios i) e ii) e todas partições possíveis das suas $m = 3$ linhas são

M_1	M_2	M_1	M_2
$\emptyset$	$\{1,2,3\}$	$\{1,2\}$	$\{3\}$
$\{1\}$	$\{2,3\}$	$\{1,3\}$	$\{2\}$
$\{2\}$	$\{1,3\}$	$\{2,3\}$	$\{1\}$
$\{3\}$	$\{1,2\}$	$\emptyset$	$\{1,2,3\}$

Obviamente, por simetria, temos que considerar somente a primeira metade das possibilidades. Logo em geral um teste exaustivo do critério iii) tem que considerar 2^{m-1} partições. ◊

Observação 7.3

O critério ii) permite somente 6 tipos de colunas, caracterizados pelos coeficientes diferentes de 0: dois coeficientes 1, ou dois coeficientes -1, ou um coeficiente 1 e outro -1, ou somente um coeficiente 1, ou -1, ou completamente 0.

$$\begin{pmatrix} 1 & -1 & 1 & 1 & -1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & -1 & -1 & 0 & 0 & 0 \end{pmatrix}$$

Os coeficientes podem ocorrer em qualquer linha. Somente os primeiros três tipos precisam satisfazer o critério iii). Eles restringem as partições possíveis: as linhas dos coeficientes de uma coluna do tipo $\binom{1}{1}$ ou $\binom{-1}{-1}$ tem que ficar em partes diferentes, aqueles de uma coluna do tipo $\binom{-1}{1}$ no mesmo parte. ◊

Exemplo 7.3 (Matriz TU pelo critério de linhas)
A matriz

$$\begin{pmatrix} 1 & -1 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{pmatrix}$$

satisfaz o critério i), porque tem coeficientes em $\{-1, 0, 1\}$, o critério ii) porque cada coluna tem no máximo dois coeficientes não-nulos, e o critério iii) com $M_1 = [1, 3], M_2 = \emptyset$. ◊

Exemplo 7.4 (Matriz TU, mas o critério de partição de linhas não se aplica)
A matriz

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

é TU (ver exemplo 7.5) mas a regra de partição de linhas não se aplica! ◊

Uma caracterização (i.e. um critério necessário e suficiente) das matrizes totalmente unimodulares é

Teorema 7.1 (Ghouila-Houri (1962))
Um matriz $A \in \mathbb{Z}^{m \times n}$ é TU sse para todo subconjunto $R \subseteq [m]$ de linhas existe uma partição $R_1 \dot\cup R_2$ tal que

$$\left| \sum_{i \in R_1} a_{ij} - \sum_{i \in R_2} a_{ij} \right| \leq 1 \tag{7.1}$$

para todas colunas $j \in [n]$.

Observe que a proposição 7.2 implica o critério acima: dado uma partição das linhas de acordo com 7.2, para todo $R \subseteq [m]$, a partição $(M_1 \cap R) \dot\cup (M_2 \cap R)$ satisfaz (7.1).

Definição 7.2
Uma matriz $A \in \mathbb{B}^{m\times n}$ possui a propriedade de *uns consecutivos* se para cada coluna $j \in [n]$, $a_{ij} = 1$ e $a_{i'j} = 1$ com $i < i'$ implica $a_{kj} = 1$ para $k \in [i, i']$.

Uma aplicação do critério de Ghouila-Houri é a

Proposição 7.3
Uma matriz que satisfaz a propriedade de uns consecutivos é totalmente unimodular.

Prova. A matriz formada por um subconjunto de linhas $R \subseteq [m]$ também possui a propriedade de uns consecutivos. Seja $R = \{i_1, \ldots, i_k\}$ com $i_1 \leq \cdots \leq i_k$. A partição em $M_1 = \{i_1, i_3, \ldots\}$ e $M_2 = \{i_2, i_4, \ldots\}$ satisfaz (7.1). ■

Exemplo 7.5
A matriz

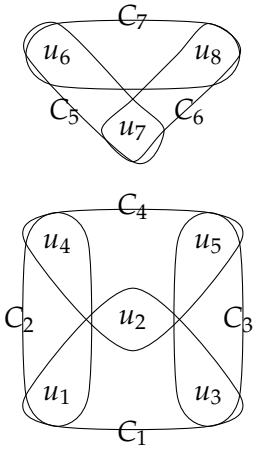
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

do exemplo 7.4 satisfaz a propriedade de uns consecutivos. Logo ela é TU. ◊

Exemplo 7.6
Para um universo $U = \{u_1, \ldots, u_m\}$, e uma família de conjuntos $C_1, \ldots, C_n \subseteq U$ com pesos $p_1, \ldots, p_n$ uma cobertura é uma seleção de conjuntos $S \subseteq [n]$ tal que cada elemento do universo é coberto, i.e. para todo $u \in U$ existe um $i \in S$ com $u \in C_i$. O problema de encontrar a cobertura de menor peso total pode ser formulado por

$$\begin{aligned} \textbf{minimiza} \quad & \sum_{i\in[n]} p_i x_i \\ \textbf{sujeito a} \quad & Ax \geq 1, \\ & x \in \mathbb{B}^n. \end{aligned}$$

com $a_{ij} = 1$ sse $u_i \in C_j$. (Figura 7.1 mostra um exemplo de uma instância e a matriz A correspondente.) Este problema em geral é NP-completo. Pela propriedade de uns consecutivos, podemos ver que no caso de um universo $U = [m]$ com subconjuntos que são intervalos o problema pode ser resolvido em tempo polinomial. ◊



$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Figura 7.1.: Exemplo de uma instância do problema de cobertura por conjuntos e a matriz A da formulação inteira correspondente.

Consequências

Teorema 7.2 (Hoffman e Kruskal (1956))
Se a matriz A de um programa linear é totalmente unimodular e o vetor b é inteiro, todas soluções básicas são inteiras. Em particular as regiões

$$\{x \in \mathbb{R}^n \mid Ax \leq b\}$$
$$\{x \in \mathbb{R}^n \mid Ax \geq b\}$$
$$\{x \in \mathbb{R}^n \mid Ax \leq b, x \geq 0\}$$
$$\{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$$

possuem pontos extremos inteiros.

Prova. Considerações acima. ■

Exemplo 7.7 (Caminhos mais curtos)

Exemplo: Caminhos mais curtos

- Dado um grafo direcionado $G = (V, A)$ com custos $c : A \to \mathbb{Z}$ nos arcos.
- Qual o caminho mais curto entre dois nós $s, t \in V$?

Exemplo: Caminhos mais curtos

$$\begin{aligned}
\textbf{minimiza} \quad & \sum_{a \in A} c_a x_a \\
\textbf{sujeito a} \quad & \sum_{a \in N^+(s)} x_a - \sum_{a \in N^-(s)} x_a = 1, \\
& \sum_{a \in N^+(v)} x_a - \sum_{a \in N^-(v)} x_a = 0, && \forall v \in V \setminus \{s, t\}, \\
& \sum_{a \in N^+(t)} x_a - \sum_{a \in N^-(t)} x_a = -1, \\
& x_a \in \mathbb{B}, && \forall a \in A.
\end{aligned}$$

A matriz do sistema acima de forma explicita:

$$\begin{array}{c} s \\ \\ \vdots \\ \\ t \end{array} \begin{pmatrix} 1 & \cdots & & \cdots & -1 \\ & & 1 & & \\ \vdots & & & & \\ & & -1 & & 1 \\ -1 & \cdots & & & \end{pmatrix} \begin{pmatrix} x_{a_1} \\ \\ \vdots \\ \\ x_{a_m} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{pmatrix}$$

Como cada arco é incidente a dois vértices, cada coluna contém um coeficiente 1 e -1, e a Proposição 7.2 é satisfeito pela partição trivial $\emptyset \dot{\cup} V$. ◊

Exemplo 7.8 (Fluxo em redes)

Exemplo: Fluxo em redes

- Dado: Um grafo direcionado $G = (V, A)$
  - com arcos de capacidade limitada $l : A \to \mathbb{Z}^+$,
  - demandas $d : V \to \mathbb{Z}$ dos vértices,
  - (com $d_v < 0$ para destino e $d_v > 0$ nos fonte)
  - e custos $c : A \to \mathbb{R}$ por unidade de fluxo nos arcos.
- Qual o fluxo com custo mínimo?

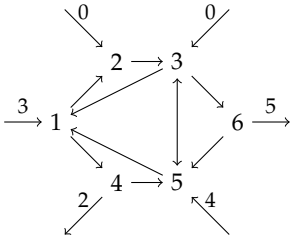


Figura 7.2.: Exemplo de uma instância de um problema de fluxo.

Exemplo: Fluxo em redes

$$\begin{aligned} \textbf{minimiza} \quad & \sum_{a \in A} c_a x_a \\ \textbf{sujeito a} \quad & \sum_{a \in N^+(v)} x_a - \sum_{a \in N^-(v)} x_a = d_v, && \forall v \in V \\ & 0 \leq x_a \leq l_a, && \forall a \in A. \end{aligned}$$

com conjunto de arcos entrantes $N^-(v)$ e arcos saintes $N^+(v)$.

Exemplo: Fluxo

- A matriz que define um problema de fluxo é totalmente unimodular.
- Consequências
  - Cada ponto extremo da região víavel é inteira.
  - A relaxação PL resolve o problema.
- Existem vários subproblemas de fluxo mínimo que podem ser resolvidos também, p.ex. fluxo máximo entre dois vértices.

◊

Exemplo 7.9 (Emparelhamentos)

EMPARELHAMENTO MÁXIMO (EM)

Entrada Um grafo $G = (V, E)$ não-direcionado.

Solução Um emparelhamento $M \subseteq E$, i.e. um conjunto de arcos, tal que para todos vértices v temos $|N(v) \cap M| \leq 1$.

Objetivo Maximiza $|M|$.

Uma formulação é

$$\textbf{maximiza} \quad \sum_{e \in E} c_e x_e \tag{7.2}$$

$$\textbf{sujeito a} \quad \sum_{u \in N(v)} x_{uv} \leq 1, \qquad \forall v \in V, \tag{7.3}$$

$$x_e \in \mathbb{B}.$$

A matriz de coeficientes dessa formulação é TU para grafos bipartidos. Por quê? Isso ainda é válido para grafos não-bipartidos? ◊

7.3. Desigualdades válidas

Desigualdades válidas

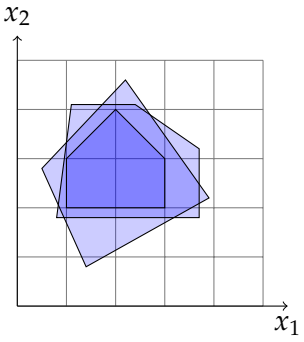


Figura 7.3.: Diferentes formulações dos mesmo PI.

- Problema inteiro
$$\max\{c^t x \mid Ax \leq b, x \in \mathbb{Z}^n_+\}$$
- Relaxação linear
$$\max\{c^t x \mid Ax \leq b, x \in R^n_+\}$$

Desigualdades válidas

Definição 7.3
Uma desigualdade $\pi x \leq \pi_0$ é *válida* para um conjunto P, se $\forall x \in P : \pi x \leq \pi_0$.

- Como encontrar desigualdades (restrições) válidas para o conjunto da soluções viáveis $\{x \mid Ax \leq b, x \in \mathbb{Z}^n_+\}$ de um problema inteiro?
  - Técnicas de construção (p.ex. método de Chvátal-Gomory)
  - Observar e formalizar características específicas do problema.
  - "The determination of families of strong valid inequalities is more of an art than a formal methodology" Nemhauser e Wolsey (1999, p. 259)

Exemplo 7.10 (Localização de facilidades não-capacitado)
Temos um conjunto de cidades $C = [n]$ em que podemos abrir facilidades para um custo fixo $f_j, j \in C$. Em cada cidade i existe um demanda que pode ser satisfeito por uma facilidade na cidade j com custo c_{ij}, caso existe um facilidade na cidade j. Com $x_{ij} \in B$ indicando que a demanda da cidade i é satisfeito pela facilidade na cidade j podemos formular

$$\textbf{minimiza} \quad \sum_{j\in[n]} f_j y_j + \sum_{i\in[n],j\in[n]} c_{ij} x_{ij} \tag{7.4}$$

$$\textbf{sujeito a} \quad \sum_{j\in[n]} x_{ij} = 1, \qquad \forall i \in [n], \tag{7.5}$$

$$x_{ij} \leq y_j, \qquad \forall i \in [n], j \in [n], \tag{7.6}$$

$$x_{ij} \in \mathbb{B}, \qquad \forall i \in [n], j \in [n], \tag{7.7}$$

$$y_j \in \mathbb{B}, \qquad \forall j \in [n]. \tag{7.8}$$

Ao invés de

$$x_{ij} \leq y_j \tag{7.9}$$

podemos formular

$$\sum_{i \in [n]} x_{ij} \leq n y_j. \tag{7.10}$$

Essa formulação ainda é correta, mas usa n restrições ao invés de n^2. Entretanto, a qualidade da relação linear é diferente. É simples ver que podemos obter (7.10) somando (7.9) sobre todos i. Portanto, qualquer solução que satisfaz (7.9) satisfaz (7.10) também, e dizemos que (7.9) *domina* (7.10).
O seguinte exemplo mostra que o contrário não é verdadeiro. Com custos de instalação $f_j = 1$, de transporte $c_{ij} = 5$ para $i \neq j$ e $c_{ii} = 0$, duas cidades e uma fábrica obtemos as duas formulações (sem restrições de integralidade)

minimiza	$y_1 + y_2 + 5x_{12} + 5x_{21}$,	$y_1 + y_2 + 5x_{12} + 5x_{21}$
sujeito a	$x_{11} + x_{12} = 1$,	$x_{11} + x_{12} = 1$,
	$x_{21} + x_{22} = 1$,	$x_{21} + x_{22} = 1$,
	$y_1 + y_2 \leq 1$,	$y_1 + y_2 \leq 1$,
	$x_{11} \leq y_1$,	$x_{11} + x_{21} \leq 2y_1$,
	$x_{12} \leq y_2$,	
	$x_{21} \leq y_1$,	$x_{21} + x_{22} \leq 2y_2$.
	$x_{22} \leq y_2$.	

A solução ótima do primeiro sistema é $y_1 = 1, x_{11} = x_{21} = 1$ com valor 6, que é a solução ótima inteira. Do outro lado, a solução ótima da segunda formulação é $y_1 = y_2 = 0.5$ com $x_{11} = x_{22} = 1$, com valor 1, i.e. ficam instaladas duas "meia-fábricas" nas duas cidades! ◊

Exemplo 7.11 (Problema do caixeiro viajante)
Na introdução discutimos a formulação básica do PCV

minimiza	$\sum_{i,j \in N} c_{ij} x_{ij}$,		
sujeito a	$\sum_{j \in N} x_{ij} = 1$,	$\forall i \in N$,	(7.11)
	$\sum_{j \in N} x_{ji} = 1$,	$\forall i \in N$,	(7.12)
	$x_{ij} \in \mathbb{B}$,	$\forall i, j \in N$,	(7.13)
	+ restrições de eliminação de subciclos!		

Uma ideia de eliminar subciclos é a seguinte: considere um subconjunto $S \subset N$ de cidades: entre cidades em S não podemos selecionar mais que $|S| - 1$ arestas, senão vai formar um subciclo. Logo uma forma de eliminar subciclos é pelas restrições

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1, \qquad \forall S \subseteq N, S \neq \emptyset, S \neq N. \tag{S_1}$$

Uma outra forma pode ser obtida como segue: associa um "potencial" (uma altura) p_i a cada cidade $i \in N$ e força o sucessor de i na rota ter um potencial pelo menos $p_i + 1$. Isso não tem como satisfazer em ciclos. Para permitir um ciclo global, vamos excluir uma cidade fixa $s \in S$ dessa restrição. Logo, as restrições

$$p_i + n(x_{ij} - 1) + 1 \leq p_j, \qquad \forall i, j \in N \setminus \{s\}, \tag{S_2}$$

também eliminam os subciclos.
Quais restrições são melhores? Considere as soluções

$$P_{S_1} = \{x \mid x \text{ satisfaz } (7.11), (7.12), (7.13), (S_1)\}$$

da primeira formulação e as soluções

$$P_{S_2} = \{x \mid \text{existem valores } p \text{ tal que } x \text{ satisfaz } (7.11), (7.12), (7.13), (S_2)\}$$

da segunda. Não é difícil de ver que existem soluções fracionárias $x \in P_{S_2}$ que não pertencem a P_{S_1}: um exemplo é dado na Figura 7.4.
É possível mostrar que $P_{S_1} \subset P_{S_2}$. Logo a formulação (S_1) domina a formulação (S_2).

$\Diamond$

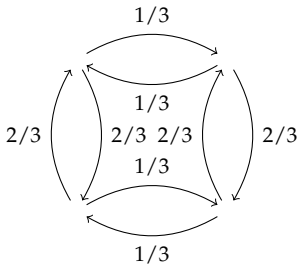
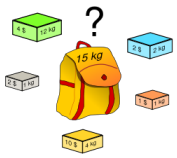


Figura 7.4.: Uma solução fracionária de uma instância do PCV com 4 cidades da formulação P_{S_2} que não é válida na formulação P_{S_1}. O valor $p_i = 0$ para todos $i \in N$.

Exemplo: 0-1-**Mochila**

$$\begin{aligned} \textbf{maximiza} \quad & \sum_{i \in [n]} v_i x_i \\ \textbf{sujeito a} \quad & \sum_{i \in [n]} p_i x_i \leq P, \\ & x_i \in \mathbb{B}. \end{aligned}$$



Exemplo: $79x_1 + 53x_2 + 53x_3 + 45x_4 + 45x_5 \leq 178$.

Exemplo 7.12 (Problema da mochila)

Exemplo: 0-1-**Mochila**

- Observação: Para um subconjunto $S \subset [1, n]$:
  Se $\sum_{i \in S} p_i > P$ então $\sum_S x_i \leq |S| - 1$.
- Exemplos:

$$\begin{aligned} x_1 + x_2 + x_3 &\leq 2, \\ x_1 + x_2 + x_4 + x_5 &\leq 3, \\ x_1 + x_3 + x_4 + x_5 &\leq 3, \\ x_2 + x_3 + x_4 + x_5 &\leq 3. \end{aligned}$$

Um conjunto S tal $\sum_{i \in S} p_i > P$ se chama uma *cobertura* e a desigualdades obtidos por tais conjuntos *desigualdades de cobertura* (ingl. cover inequalities).

◊

Exemplo 7.13 (Emparelhamentos)
Continuando exemplo 7.9.

Exemplo: Emparelhamentos

- Escolhe um subconjunto arbitrário de vértices $U \subseteq V$.
- Observação: O número de arestas internas é $\leq \lfloor |U|/2 \rfloor$.
- Portanto:

$$\sum_{a \in U^2 \cap A} x_a \leq \lfloor |U|/2 \rfloor \tag{7.14}$$

  é uma desigualdade válida.

◊

Observação 7.4
A envoltória convexa do problema de emparelhamentos é dado pelas restrições (7.3) e (7.14) para todo conjunto U de cardinalidade impar maior que 1.

◊

Método de Chvátal-Gomory
Dado uma restrição

$$\sum_{i\in[n]} a_i x_i \leq b$$

também temos, para $u \in \mathbb{R}, u > 0$ as restrições válidas

$$\sum_{i\in[n]} u a_i x_i \leq ub \qquad \text{(multiplicação com } u\text{)}$$

$$\sum_{i\in[n]} \lfloor u a_i \rfloor x_i \leq ub \qquad \text{porque } \lfloor y \rfloor \leq y \text{ e } 0 \leq x_i$$

$$\sum_{i\in[n]} \lfloor u a_i \rfloor x_i \leq \lfloor ub \rfloor \qquad \text{porque o lado da esquerda é inteira}$$

O método de Chvátal-Gomory funciona igualmente para combinações lineares de colunas. Com $A = (a^1\ a^2\ \cdots a^n)$ e $u \in \mathbb{R}^m$ obtemos

$$\sum_{i\in[n]} \left\lfloor u^t a^i \right\rfloor x_i \leq \lfloor u^t b \rfloor \tag{7.15}$$

Teorema 7.3
Cada desigualdade válida pode ser construída através de um número finito de aplicações do método de Chvátal-Gomory (7.15).

(Uma prova do teorema encontra-se, por exemplo, em Nemhauser e Wolsey (1999, p. II.1.2) ou, para o caso de variáveis 0-1, em Wolsey (1998, Th. 8.4).)

Observação 7.5
Para desigualdades $\sum_{i\in[n]} a_i x_i \geq b$ obtemos similarmente

$$\sum_{i\in[n]} \left\lceil u^t a^i \right\rceil x_i \geq \lceil u^t b \rceil$$

◊

Exemplo 7.14 (Problema da mochila)
A relaxação linear do problema da mochila acima possui as restrições

$$\begin{array}{rrrrrcl}
79x_1 & +53x_2 & +53x_3 & +45x_4 & +45x_5 & \leq & 178, \\
x_1 & & & & & \leq & 1, \\
& x_2 & & & & \leq & 1, \\
& & x_3 & & & \leq & 1, \\
& & & x_4 & & \leq & 1, \\
& & & & x_5 & \leq & 1,
\end{array}$$

Com $u = (1/79\ 0\ 26/79\ 26/79\ 0\ 0)^t$ obtemos a desigualdade válida

$$x_1 + x_2 + x_3 \leq 2.$$

$\Diamond$

Exemplo 7.15 (Emparelhamentos)
Aplicando o método de Chvátal-Gomory para $U \subseteq V$ com $u = (1/2\ 1/2 \cdots 1/2)^t \in \mathbb{R}^{|U|}$ às desigualdades

$$\sum_{u \in N(v)} x_{uv} \leq 1, \qquad \forall v \in U$$

para obter

$$\sum_{v \in U} 1/2 \sum_{u \in N(v)} x_{uv} = \sum_{a \in U^2 \cap A} x_a + \sum_{a \in N(U)} 1/2 x_a \leq |U|/2$$

e depois aplicar os pisos com $\sum_{a \in N(U)} \lfloor 1/2 \rfloor\, x_a = 0$

$$\sum_{a \in U^2 \cap A} x_a \leq \lfloor |U|/2 \rfloor\,.$$

$\Diamond$

7.4. Planos de corte

Como usar restrições válidas?

- Adicionar à formulação antes de resolver.
  - Vantagens: Resolução com ferramentas padrão.
  - Desvantagens: Número de restrições pode ser muito grande ou demais.
- Adicionar ao problema se necessário: Algoritmos de plano de corte.
  - Vantagens: Somente cortes que ajudam na solução da instância são usados.

Planos de corte

Problema inteiro

$$\max\{c^t x \mid Ax \leq b, x \in \mathbb{Z}_+^n\}$$

- O que fazer, caso a relaxação linear não produz soluções ótimas?
- Um método: Introduzir *planos de corte*.

  Definição 7.4
  Um plano de corte (ingl. cutting plane) é uma restrição válida (ingl. valid inequality) que todas soluções inteiras satisfazem.

Algoritmo de planos de corte

Algoritmo 7.1 (Planos de corte)
Entrada Programa inteiro $\max\{c^t x \mid Ax \leq b, x \in \mathbb{Z}_+^n\}$.

Saida Solução inteira ótima.

$V := \{x \mid Ax \leq b\}$ `{ região viável }`
$x^* := \text{argmax}\{c^t x \mid x \in V\}$ `{ resolve relaxação }`
`while` $(x^* \notin \mathbb{Z}_+^n)$ `do`
  `encontra um corte` $a^t x \leq d$ `tal que` $a^t x^* > d$
  $V := V \cap \{x \mid a^t x \leq d\}$ `{ nova região viável }`
  $x^* := \text{argmax}\{c^t x \mid x \in V\}$ `{ nova solução ótima }`
`end while`

Método de Gomory

- Podemos garantir que sempre existe um novo corte na linha 4? Como achar esse novo corte?
- A solução ótima atual é representado pelo dicionário

$$z = \bar{z} + \sum_j \bar{c}_j x_j$$
$$x_i = \bar{b}_i - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j \qquad i \in \mathcal{B}$$

- Se a solução não é inteira, existe um índice i tal que $x_i \notin \mathbb{Z}_+$, i.e. $\bar{b}_i \notin \mathbb{Z}_+$.

Cortes de Chvátal-Gomory

$$x_i = \bar{b}_i - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j \qquad \text{Linha fracionária} \qquad (7.16)$$

$$x_i \leq \bar{b}_i - \sum_{j \in \mathcal{N}} \lfloor \bar{a}_{ij} \rfloor x_j \qquad \text{Definição de } \lfloor \cdot \rfloor \qquad (7.17)$$

$$x_i \leq \lfloor \bar{b}_i \rfloor - \sum_{j \in \mathcal{N}} \lfloor \bar{a}_{ij} \rfloor x_j \qquad \text{Integralidade de } x \qquad (7.18)$$

$$0 \geq \{\bar{b}_i\} - \sum_{j \in \mathcal{N}} \{\bar{a}_{ij}\} x_j \qquad (7.16) - (7.18) \qquad (7.19)$$

$$x_{n+1} = -\{\bar{b}_i\} + \sum_{j \in \mathcal{N}} \{\bar{a}_{ij}\} x_j \qquad \text{Nova variável} \qquad (7.20)$$

$$x_{n+1} \in \mathbb{Z}_+ \qquad (7.21)$$

Para soluções inteiras, a diferença do lado esquerdo e do lado direito na equação (7.18) é inteira. Como uma solução inteira também satisfaz a equação (7.16) podemos concluir que x_{n+1} também é inteira.

Observação 7.6
Lembra que o parte fracionário de um número é definido por $\{x\} = x - \lfloor x \rfloor$, sendo o piso $\lfloor x \rfloor$ o maior número inteiro menor que x. Por exemplo, $\{0.25\} = 0.25$ e $\{-0.25\} = 0.75$. (Ver definição A.1 na página 197.) ◊

A solução básica atual não satisfaz (7.19), porque com $x_j = 0, j \in \mathcal{N}$ temos que satisfazer

$$\{\bar{b}_i\} \leq 0,$$

uma contradição com a definição de $\{\cdot\}$ e o fato que $\bar{b}_i$ é fracionário. Portanto, provamos

Proposição 7.4
O corte (7.19) satisfaz os critérios da linha 4 do algoritmo Planos de corte.

Exemplo 7.16
Queremos resolver o problema

$$\begin{aligned} \textbf{maximiza} \quad & x_1 + x_2 \\ \textbf{sujeito a} \quad & -x_1 + 3x_2 \leq 9, \\ & 10x_1 \leq 27, \\ & x_1, x_2 \in \mathbb{Z}_+. \end{aligned}$$

A solução da relaxação linear produz a série de dicionários

(1) $z = x_1 + x_2$
$w_1 = 9 + x_1 - 3\mathbf{x_2}$
$w_2 = 27 - 10x_1$

(2) $z = 3 + 4/3x_1 - 1/3w_1$
$x_2 = 3 + 1/3x_1 - 1/3w_1$
$w_2 = 27 - 10\mathbf{x_1}$

(3) $z = 6.6 - 4/30w_2 - 1/3w_1$
$x_2 = 3.9 - 1/30w_2 - 1/3w_1$
$x_1 = 2.7 - 1/10w_2$

A solução ótima $x_1 = 2.7$, $x_2 = 3.9$ é fracionária. Correspondendo com a segunda linha

$x_2 = 3.9 - 1/30w_2 - 1/3w_1$

temos o corte

$w_3 = -0.9 + 1/30w_2 + 1/3w_1$

e o novo sistema é

(4) $z = 6.6 - 4/30w_2 - 1/3w_1$
$x_2 = 3.9 - 1/30w_2 - 1/3w_1$
$x_1 = 2.7 - 1/10w_2$
$w_3 = -0.9 + 1/30w_2 + 1/3\mathbf{w_1}$

Substituindo w_2 e w_1 no corte $w_3 = -0.9 + 1/30w_2 + 1/3w_1 \geq 0$ podemos reescrever o corte sando as variáveis originais do sistema, obtendo $x_2 \leq 3$. Esse sistema não é mais ótimo, e temos que re-otimizar. Pior, a solução básica atual não é viável! Mas como na função objetivo todos coeficientes ainda são negativos, podemos aplicar o método Simplex dual. Um pivô dual gera a nova solução ótima

(5) $z = 5.7 - 1/10w_2 - w_3$
$x_2 = 3 - w_3$
$x_1 = 2.7 - 1/10w_2$
$w_1 = 2.7 - 1/10w_2 + 3w_3$

com $x_2 = 3$ inteiro agora, mas x_1 ainda fracionário. O próximo corte, que corresponde com x_1 é

(6) $z = 5.7 - 1/10w_2 - w_3$
$x_2 = 3 - w_3$
$x_1 = 2.7 - 1/10w_2$
$w_1 = 2.7 - 1/10w_2 + 3w_3$
$w_4 = -0.7 + \mathbf{1/10w_2}$

(7) $z = 5 - w_4 - w_3$
$x_2 = 3 - w_3$
$x_1 = 2 - w_4$
$w_1 = 2 - w_4 + 3w_3$
$w_2 = 7 + 10w_4$

cuja solução é inteira e ótima. (O último corte inserido $w_4 = -0.7 + 1/10w_2 \geq 0$ corresponde com $x_1 \leq 2$.) ♢

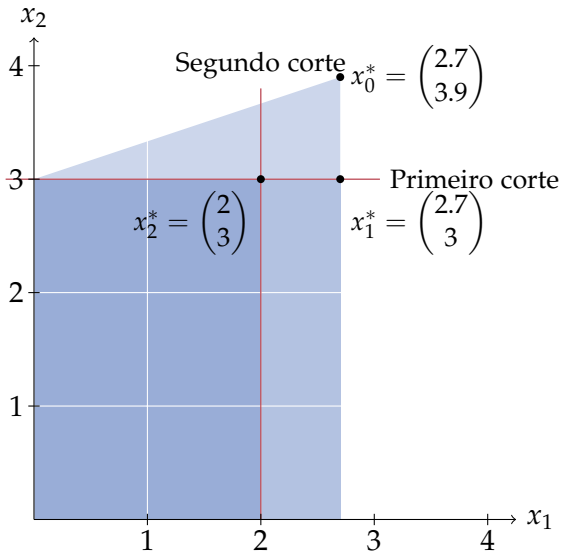


Figura 7.5.: Visualização do exemplo 7.16.

Observação 7.7
Nosso método se aplica somente para sistemas *puros* (ver página 115) e temos que garantir que as variáveis de folga são variáveis inteiras. Por isso os coeficientes de um sistema original em forma normal tem que ser números inteiros, i.e., $A \in \mathbb{Z}^{n \times m}$ e $b \in \mathbb{Z}^m$. ♢

Resumo: Algoritmos de planos de corte

- O algoritmo de planos de corte, usando os cortes de Gomory termina sempre, i.e. é correto.
- O algoritmos pode ser modificado para programas mistos.
- A técnica é considerado inferior ao algoritmos de branch-and-bound.
- Mas: Planos de corte em combinação com branch-and-bound é uma técnica poderosa: Branch-and-cut.

7.5. Algoritmos Branch-and-bound

Branch-and-bound
Ramifica-e-limite (ingl. branch-and-bound, Land e Doig (1960))

- Técnica geral para problemas combinatoriais.

  > Branch and Bound is by far the most widely used tool for solving large scale NP-hard combinatorial optimization problems. (Clausen 1999)

- Ideia básica:
  - Particiona um problema em subproblemas disjuntos e procura soluções recursivamente.
  - Evite percorrer toda árvore de busca, calculando limites e cortando sub-árvores.
- Particularmente efetivo para programas inteiras: a relaxação linear fornece os limites.

Limitar

- Para cada sub-árvore mantemos um limite inferior e um limite superior.
  - Limite inferior: Valor da melhor solução encontrada na sub-árvore.
  - Limite superior: Estimativa (p.ex. valor da relaxação linear na PI)
- Observação: A eficiência do método depende crucialmente da qualidade do limite superior.

Cortar sub-árvores

Podemos cortar ...

(1) por inviabilidade: Sub-problema é inviável.

(2) por limite: Limite superior da sub-árvore $\overline{z}_i$ menor que limite inferior global $\underline{z}$ (o valor da melhor solução encontrada).

(3) por otimalidade: Limite superior $\overline{z}_i$ igual limite inferior $\underline{z_i}$ da sub-árvore.

Observação: Como os cortes dependem do limite $\underline{z}$, uma boa solução inicial pode reduzir a busca consideravelmente.

Ramificar

- Não tem como cortar mais? Escolhe um nó e particiona.
- Qual a melhor ordem de busca?
- Busca por profundidade
  - V: Limite superior encontrado mais rápido.
  - V: Pouca memória ($O(\delta d)$, para δ subproblemas e profundidade d).
  - V: Re-otimização eficiente do pai (método Simplex dual)
  - D: Custo alto, se solução ótima encontrada tarde.
- Melhor solução primeiro ("best-bound rule")
  - V: Procura ramos com maior potencial.
  - V: Depois encontrar solução ótima, não produz ramificações supérfluas.
- Busca por largura? Demanda de memória é impraticável.

Em resumo: um algoritmo de branch-and-bound consiste de quatro componentes principais:

- Uma heurística que encontra uma boa solução inicial;
- um limite inferior (no caso de minimização) ou superior (para maximização) do valor de um subproblema;
- uma estratégia de ramificação, que decompõe um problema em subproblemas;
- uma estratégia de seleção, que escolhe o próximo subproblema entre os subproblemas ativos.

Algoritmos B&B

Algoritmo 7.2 (B&B)
Instância Programa inteiro $P = \max\{c^t x \mid Ax \leq b, x \in Z_+^n\}$.

Saida Solução inteira ótima.

```
{ com z̄(P) um limite superior para problema P }
z := −∞          { limite inferior }
A := {(P, z̄(P))} { ns ativos }
while A ≠ ∅ do
  Escolhe: (P, z̄(P)) ∈ A; A := A \ (P, z̄(P))
  Ramifique: Gera subproblemas P_1, ..., P_n.
  for all P_i, 1 ≤ i ≤ n do
    { adiciona, se permite melhor soluo }
    if z̄(P_i) > z then
      A := A ∪ {(P_i, z̄(P_i))}
    end if
    { atualize melhor soluo }
    if (soluo z̄(P_i) vivel) then
      z := z̄(P_i)
    end if
  end for
end while
```

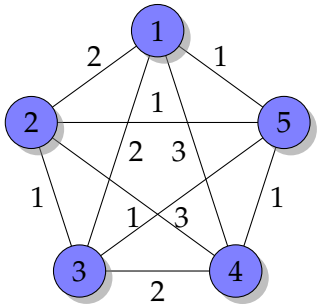
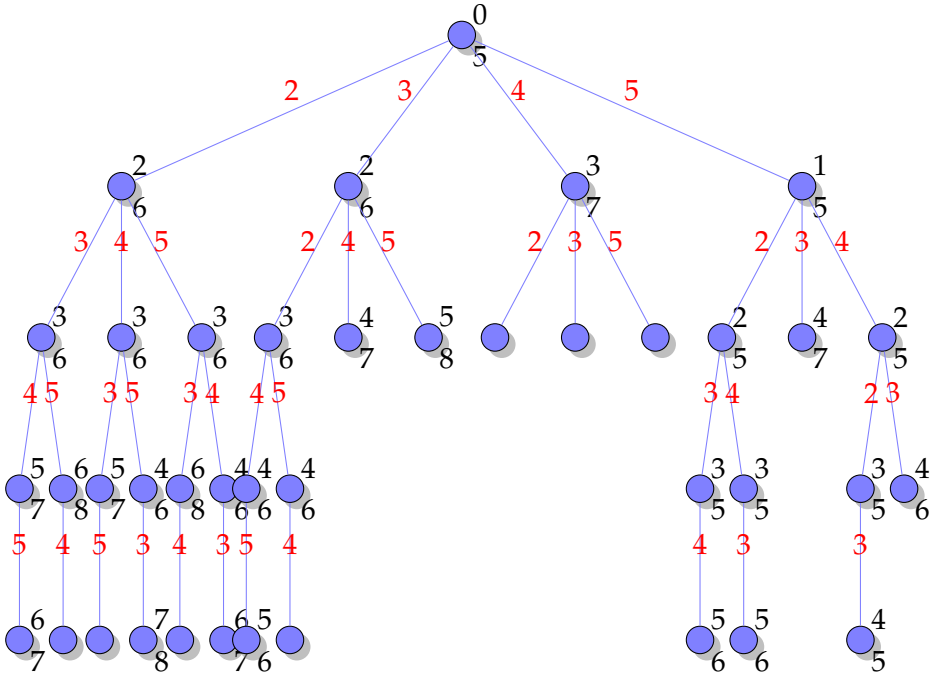


Figura 7.6.: Exemplo de uma instância do PCV.

Exemplo 7.17 (Aplicação Branch-and-Bound no PCV)
Considera uma aplicação do PCV no grafo da Figura 7.6.
Aplicando somente backtracking obtemos a seguinte árvore de busca:



A árvore de backtracking completa possui 65 vértices (por nível: 1,4,12,24,24). Usando como limite inferior o custo atual mais o número de arcos que faltam vezes a distância mínima e aplicando branch-and-bound obtemos os custos parciais e limites indicados na direita de cada vértice. Com isso podemos aplicar uma série de cortes: busca da esquerda para direito obtemos

- uma nova solução 7 em 2345;
- um corte por limite em 235;
- um corte por otimalidade em 243;
- um corte por otimalidade em 2453;
- um corte por limite em 253;
- um corte por otimalidade em 2543;
- uma nova solução 6 em 3245;
- um corte por otimalidade em 32;
- um corte por otimalidade em 3;
- um corte por limite em 4;
- um corte por otimalidade em 5234;
- um corte por otimalidade 5243;
- um corte por limite em 53;
- um corte por otimalidade 543.

$\Diamond$

Exemplo 7.18 (Escalonamento de tarefas)
Considera o problema de escalonamento $1 \mid r_j \mid L_{\max}$: temos n tarefas a serem executadas numa única máquina. Cada tarefa possui um tempo de execução p_j e é disponível a partir do tempo r_j (release date) e idealmente tem que terminar antes do prazo d_j (due date). Caso a tarefa j termina no tempo C_j o seu atraso é $L_j = \max\{0, C_j - d_j\}$. Uma tarefa tem que ser executada sem interrupção. Queremos encontrar uma sequenciamento das tarefas tal que o atraso máximo é minimizado. (Observe que uma solução é uma permutação das tarefas.)
Um exemplo de uma instância com quatro tarefas é

Tarefa	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	11

Uma abordagem via branch-and-bound é explorar todas permutações possíveis. Um limite inferior bom para a função objetivo pode ser obtido como segue: o problema sem *release dates* $1 \mid\mid L_{\max}$ possui uma solução simples polinomial, conhecida como EDD (earliest due date): ordene as tarefas por *due date*. No nosso caso é possível que durante a execução de uma tarefa passamos o release de uma outra tarefa com *due date* menor. Para considerar isso, o nosso limite inferior será o sequenciamento obtido pela regra EDD, permitindo interrupções. ◊

Branch-and-bound e PI

- Problema PI (puro): $\{\max c^t x \mid x \in S, x \in \mathbb{Z}_+^n\}$.
- Resolve a relaxação linear.
- Solução inteira? Problema resolvido.
- Caso contrário: Escolhe uma variável inteira x_i, com valor $\bar{b}_i$ fracionário.
- Heurística: Variável mais fracionária: $\operatorname{argmin}_i |\{x_i\} - 0.5|$.
- Particione o problema $S = S_1 \dot{\cup} S_2$ tal que

$$S_1 = S \cap \{x \mid x_i \leq \lfloor v_i \rfloor\}; \quad S_2 = S \cap \{x \mid x_i \geq \lceil v_i \rceil\}$$

- Em particular com variáveis $x_i \in \mathbb{B}$:

$$S_1 = S \cap \{x \mid x_i = 0\}; \quad S_2 = S \cap \{x \mid x_i = 1\}$$

- Preferimos formulações mais "rígidas".

7.6. Notas

É possível testar se uma matriz é totalmente unimodular em tempo polinomial $O((n+m)^3)$ (Truemper 1990)[1]. Porém decidir se uma matriz possui uma submatriz que satisfaz a propriedade de uns consecutivos, ou pode ser particionado em duas matrizes com essa propriedade, bem como encontrar o menor número de alterações de uma matriz que torna-lá ter essa propriedade é NP-completo (Garey e Johnson 1979, SR14–16). Clausen (1999) dá uma boa introdução em algoritmos de branch-and-bound, com mais exemplos e exercícios. O artigo do Cook (2012) relata a história do método. Concorde atualmente é o melhor solver exato para o problema do caixeiro viajante. Exemplos de soluções e código aberto do solver é disponível na sua página web (Cook 2011). A aplicação do método branch-and-bound para PI segue Dakin (1965).

7.7. Exercícios

(Soluções a partir da página 231.)

Exercício 7.1 (Matrizes totalmente unimodulares)
Mostra que a seguinte generalização do item 2 da proposição 7.1 é válido: Para uma matriz arbitrária $A \in \{-1,0,1\}^{m\times n}$ e uma matriz $B \in \{-1,0,1\}^{m\times o}$ com no máximo um coeficiente não-nulo em cada coluna, a matriz $(A\ B)$ é totalmente unimodular sse a matriz A é totalmente unimodular.

Exercício 7.2 (Matrizes totalmente unimodulares)
Para cada um dos problemas do exercício 6.2 decide, se a matriz de coeficientes é totalmente unimodular.

Exercício 7.3 (Matrizes totalmente unimodulares)
Prove ou mostre um contra-exemplo.

a) Se A é totalmente unimodular, então $\left(\begin{smallmatrix} A & 0 \\ 0 & A \end{smallmatrix}\right)$ também.

b) Se A é totalmente unimodular, então $\left(\begin{smallmatrix} A & A^t \end{smallmatrix}\right)$ também.

c) Se A é totalmente unimodular, então $\left(\begin{smallmatrix} A & A \\ A & 0 \end{smallmatrix}\right)$ também.

[1] O problema consta como "aberto" em Garey e Johnson (1979, OPEN10).

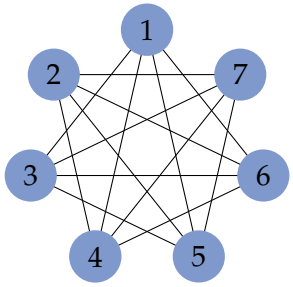


Figura 7.7.: Instância do problema do conjunto independente máximo.

Exercício 7.4 (Desigualdades válidas (Nemhauser,Wolsey))
Uma formulação do problema do conjunto independente máximo é

$$\begin{aligned} \textbf{maximiza} \quad & \sum_{v \in V} x_v, && && (7.22)\\ \textbf{sujeito a} \quad & x_u + x_v \leq 1, && \forall \{u,v\} \in E, && (7.23)\\ & x_v \in \mathbb{B}, && \forall v \in V. && (7.24) \end{aligned}$$

Considere a instância da Figura 7.7. Mostra que $\sum_{i \in [7]} x_i \leq 2$ é uma desigualdade válida.

Exercício 7.5 (Desigualdades válidas)
O exemplo 7.15 mostra como obter as desigualdades válidas do exemplo 7.13 usando cortes de Gomory. Mostra como obter as desigualdades válidas

$$\sum_{i \in S} x_i \leq |S| - 1$$

para um $S \subseteq [n]$ com $\sum_{i \in S} p_i > P$ do problema da mochila usando cortes de Gomory.

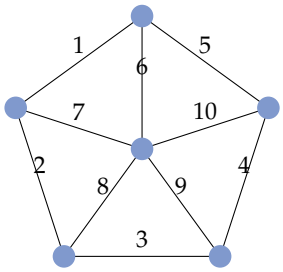


Figura 7.8.: Exemplo de uma instância do PCV.

Exercício 7.6 (Desigualdades válidas)
Considere a instância da Figura 7.8 do problema do caixeiro viajante (os números nas arestas representam os índices das variáveis correspondentes). Mostra que

$$x_1 + x_2 + x_5 + x_6 + x_7 + x_9 \leq 4$$

é uma desigualdade válida.

Exercício 7.7 (Desigualdades válidas)
Para cada uma das desigualdades válidas do exemplo 7.12 mostra como ele pode ser obtida via uma aplicação (um número finito de aplicações) do método de Chvátal-Gomory (7.15).

Exercício 7.8 (Planos de corte)
Resolve com o algoritmo de planos de corte using cortes de Chvátal-Gomory.

$$\begin{aligned} \textbf{maximiza} \quad & x_1 + 3x_2 \\ \textbf{sujeito a} \quad & -x_1 \leq -2, \\ & x_2 \leq 3, \\ & -x_1 - x_2 \leq -4, \\ & 3x_1 + x_2 \leq 12, \\ & x_i \in \mathbb{Z}_+, \end{aligned}$$

$$\begin{aligned} \textbf{maximiza} \quad & x_1 - 2x_2 \\ \textbf{sujeito a} \quad & -11x_1 + 15x_2 \leq 60, \\ & 4x_1 + 3x_2 \leq 24, \\ & 10x_1 - 5x_2 \leq 49, \\ & x_1, x_2 \in \mathbb{Z}_+, \end{aligned}$$

Exercício 7.9 (Desigualdades válidas)
Gera uma desigualdade válida similar com a desigualdade (7.15) para a restrição

$$\sum_{i \in [n]} a_i x_i \geq b.$$

8. Tópicos

Outras técnicas

- Branch-and-cut.

  Começa com menos restrições (relaxação) e insere restrições (cortes) nos sub-problemas da busca com o algoritmo branch-and-bound.

- Branch-and-price.

  Começa com menos variáveis e insere variáveis (“geração de colunas”) nos sub-problemas da busca com o algoritmo branch-and-bound.

Parte III.

Heurísticas

(Observação: isto é um capítulo antigo; sugiro consultar a notas de aula da disciplina "Técnicas de busca heurística".)

9. Introdução

Resolução de Problemas

- Problemas Polinomiais
  1. Programação Dinâmica
  2. Divisão e Conquista
  3. Algoritmos Gulosos
- Problemas Combinatórios
  - *Técnicas Exatas*: Programação Dinâmica, Divisão e Conquista backtracking, branch & bound
  - *Programação não-linear*: Programação semi-definida, etc.
  - *Algoritmos de aproximação*: garantem solução aproximada
  - *Heurísticas e metaheurísticas*: raramente provêem aproximação

Heurísticas

- O que é uma heurística?

  Practice is when it works and nobody knows why.

  Grego *heurísko*: eu acho, eu descubro.
- Qualquer procedimento que resolve um problema
  - bom em média
  - bom na prática (p.ex. Simplex)
  - não necessáriamente comprovadamente.
- Nosso foco
  - Heurísticas construtivas: Criar soluções.
  - Heurísticas de busca: Procurar soluções.

Heurísticas de Construção

- Constróem uma solução, escolhendo um elemento a ser inserido na solução a cada passo.
- Geralmente são algoritmos gulosos.
- Podem gerar soluções infactíveis.
  - Solução infactível: não satisfaz todas as restrições do problema.
  - Solução factível: satisfaz todas as restrições do problema, mas não é necessariamente ótima.

Exemplo: Heurística construtiva

- Problema do Caixeiro Viajante (PCV) – Heurística do vizinho mais próximo.

Algoritmo 9.1 (Vizinho mais próximo)
Entrada Matriz de distâncias completa $D = (d_{ij})$, número de cidades n.

Saída Uma solução factível do PCV: Ciclo Hamiltoniano C com custo c.

```
HVizMaisProx(D,n)=
  { cidade inicial aleatória }
  u := seleciona uniformemente de [1,n]
  w := u
  { representação de caminhos: sequência de vértices }
  C := u        { ciclo inicial  }
  c := 0       { custo do ciclo }
  repeat n−1 vezes
    seleciona v ∉ C com distância mínima de u
    C := C v
    c := c + d_uv
    u := v
  end repeat
  C := C w  { fechar ciclo }
  c := c + d_uw
  return (C,c)
```

Meta-heurísticas

- Heurísticas genéricas: *meta-heurísticas*.

Motivação: quando considera-se a possibilidade de usar heurísticas

- Para gerar uma solução factível num tempo pequeno, muito menor que uma solução exata pudesse ser fornecida.
- Para aumentar o desempenho de métodos exatos. Exemplo: um limitante superior de um Branch-and-Bound pode ser fornecido por uma heurística.

Desvantagens do uso de heurísticas

- No caso de metaheurísticas, não há como saber o quão distante do ótimo a solução está.
- Não há garantia de convergência.
- Dependendo do problema e instância, não há como garantir uma solução ótima.

Problema de otimização em geral

- Um problema de otimização pode ser representado por uma quádrupla

$$(I, S, f, obj)$$

  - I é o conjunto de possíveis instâncias.
  - $S(i)$ é o conjunto de soluções factíveis (espaço de soluções factíveis) para a instância i.
  - Uma função objetivo (ou *fitness*) $f(\cdot)$ avalia a qualidade de uma dada solução.
  - Um objetivo obj = min ou max: $s^* \in S$ para o qual $f(s^*)$ seja mínimo ou máximo.

- Alternativa

$$\begin{aligned} \textbf{optimiza} \quad & f(x), \\ \textbf{sujeito a} \quad & x \in S. \end{aligned}$$

- S discreto: problema combinatorial.

Técnicas de solução

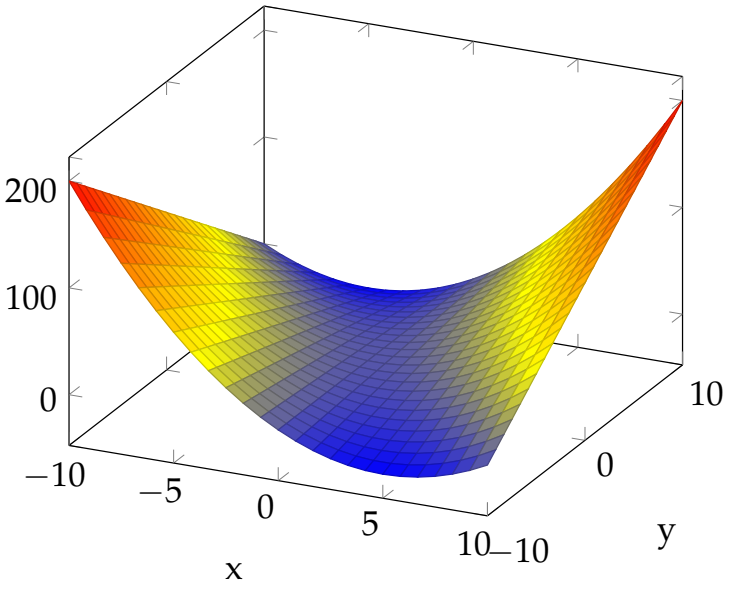
- Resolver o problema nessa geralidade: enumeração.
- Frequentemente: Uma solução $x \in S$ possui uma *estrutura*.
- Exemplo: x é uma tupla, um grafo, etc.
- Permite uma enumeração por componente: branch-and-bound.

10. Heurísticas baseadas em Busca local

10.1. Busca local

Busca Local

- Frequentemente: O espaço de soluções possui uma topologia.
- Exemplo de otimização (contínua): $\max\{x^2 + xy \mid x, y \in \mathbb{R}\}$



- Espaço Euclidiano de duas dimensões.
- Isso podemos aproveitar: Busca localmente!

Vizinhanças

- O que fazer se não existe uma topologia natural?
- Exemplo: No caso do PCV, qual o vizinho de um ciclo Hamiltoniano?
- Temos que definir uma vizinhança.

- Notação: O conjunto de soluções vizinhas de $x \in S$ é $\mathcal{N}(x)$.
- Uma vizinhança defina a *paisagem de otimização* (ingl. optimization landscape): Espaço de soluções com valor de cada solução.

Relação de vizinhança entre soluções

- Uma solução s' é obtida por uma pequena modificação na solução s.
- Enquanto que $\mathcal{S}$ e f são fornecidos pela especificação do problema, o projeto da vizinhança é livre.

Busca Local k-change e inserção

- k-change: mudança de k componentes da solução.
- Cada solução possui vizinhança de tamanho $O(n^k)$.
- Exemplo: 2-change, 3-change.
- TSP: 2-change (inversão).
- Inserção/remoção: inserção de um componente da solução, seguido da factibilização da solução
- Vertex cover: 1-change + remoção.

Exemplo: Vizinhança mais elementar

- Suponha um problema que possue como soluções factíveis $S = \mathbb{B}^n$ (por exemplo, uma instância do problema de particionamento de conjuntos).
- Então, para $n = 3$ e $s_0 = (0, 1, 0)$, para uma busca local 1-flip, $N(s_0) = \{(1, 1, 0), (0, 0, 0), (0, 1, 1)\}$.

Exemplo: Vizinhanças para TSP

- *2-exchange*: Para cada par de arcos (u_1, v_1) e (u_2, v_2) não consecutivos, remova-os da rota, e insira os arcos (u_1, u_2) e (v_1, v_2).
- Para uma solução s e uma busca k-exchange $|\mathcal{N}(s)| \in \mathrm{O}(n^k)$.

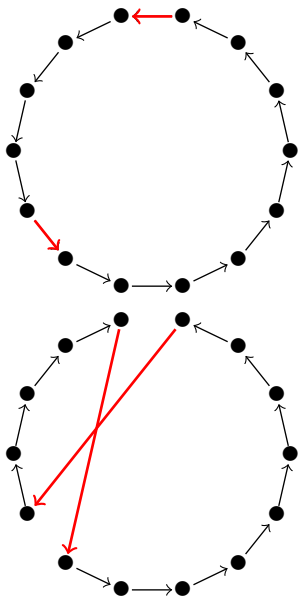


Figura 10.1.: Um movimento na vizinhança 2-exchange.

Características de vizinhanças
É desejável que uma vizinhança é

- *simétrica* (ou *reversível*)

$$y \in \mathcal{N}(x) \Rightarrow x \in \mathcal{N}(y)$$

- *conectada* (ou *completa*)

$$\forall x, y \in S : \exists z_1, \ldots, z_k \in S : \quad \begin{array}{ll} z_1 \in \mathcal{N}(x), & \\ z_{i+1} \in \mathcal{N}(z_i), & 1 \leq i < k, \\ y \in \mathcal{N}(z_k). & \end{array}$$

Busca Local: Ideia

- Inicia a partir de uma solução s_0
- Se move para soluções vizinhas melhores no espaço de busca.
- Para, se não tem soluções melhores na vizinhança.
- Mas: Repetindo uma busca local com soluções inicias randômicas, achamos o mínimo global com probabilidade 1.

Exemplo 10.1 (Método Simplex)
O método Simplex pode ser visto como busca local no espaço de vértices com uma vizinhança definido por arestas no politopo. ◊

Busca local – Caso contínuo

Algoritmo 10.1 (Busca local contínua)
Entrada Solução inicial $s_0 \in \mathbb{R}^n$, tamanho inicial α de um passo.

Saída Solução $s \in \mathbb{R}^n$ tal que $f(s) \leq f(s_0)$.

Nome Gradient descent.

```
BuscaLocal(s_0, α)=
  s := s_0
  while ∇f(s) ≠ 0 do
    s' := s − α∇f(s)
    if f(s') < f(s) then
      s := s'
    else
      diminui α
    end if
  end while
  return s
```

Busca local – Caso contínuo

- Gradiente
$$\nabla f(x) = \left(\frac{\delta f}{\delta x_1}(x), \ldots, \frac{\delta f}{\delta x_n}(x)\right)^t$$
sempre aponta na direção do crescimento mais alto de f (Cauchy).
- Necessário: A função objetivo f é diferenciável.
- Diversas técnicas para diminuir (aumentar) α.
- Opção: Line search na direção $-\nabla f(x)$ para diminuir o número de gradientes a computar.

Busca Local – Best Improvement

Algoritmo 10.2 (Busca Local BI)
Entrada Solução inicial s_0.

Saída Solução s tal que $f(s) \leq f(s_0)$.

Nomes Steepest descent, steepest ascent.

```
BuscaLocal(s_0)=
  s := s_0
  while true
    s' := argmin_y{f(y) | y ∈ N(s)}
    if f(s') < f(s) then s := s' else break
  end while
  return s
```

Busca Local – First Improvement

Algoritmo 10.3 (Busca Local FI)
Entrada Solução inicial s_0.

Saída Solução s' tal que $f(s') \leq f(s)$.

Nomes Hill descent, hill climbing.

```
BuscaLocal(s_0)=
  s := s_0
  repete
    seleciona s' ∈ N(s) no vista ainda
    if f(s') < f(s) then s := s'
  at todas solues em N(s) vistas
  returna s
```

Projeto de uma busca local

- Como gerar uma solução inicial? Aleatória, via método construtivo, etc.
- Quantas soluções inicias devem ser geradas?
- Importante: Definição da função de vizinhança $\mathcal{N}$.
- Vizinhança grande ou pequena? (grande= muito tempo e pequena=menos vizinhos)

- Estratégia de seleção de novas soluções
  - examine todas as soluções vizinhas e escolha a melhor
  - assim que uma solução melhor for encontrada, reinicie a busca. Neste caso, qual a sequência de soluções examinar?
- Importante: Método eficiente para avaliar a função objetivo de vizinhos.

Exemplo: 2-change TSP

- Vizinhança: Tamanho $O(n^2)$.
- Avaliação de uma solução: $O(n)$ (somar n distâncias).
- Atualizando a valor da solução atual: $O(1)$ (somar 4 distâncias)
- Portanto: Custo por iteração de "best improvement"
  - $O(n^3)$ sem avaliação diferential.
  - $O(n^2)$ com avaliação diferential.

Avaliação de buscas locais

Como avaliar a busca local proposta?

- Poucos resultados teóricos.
- Difícil de saber a qualidade da solução resultante.
- Depende de experimentos.

Problema Difícil

- É fácil de gerar uma solução aleatória para o TSP, bem como testar sua factibilidade.
- Isso não é verdade para todos os problemas.
- Exemplo difícil: Atribuição de pesos a uma rede OSPF.

Busca local

- Desvantagem obvia: Podemos parar em mínimos locais.
- Exceto: Função objetivo convexa (caso minimização) ou concava (caso maximização).
- Técnicas para superar isso baseadas em busca local
  - Multi-Start
  - Busca Tabu
  - Algoritmos Metropolis e Simulated Annealing
  - Variable neighborhood search

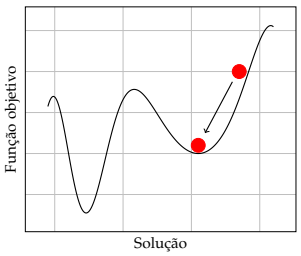


Figura 10.2.: Busca local e mínimos locais é globais.

Multi-Start Metaheuristic

- Gera uma solução aleatória inicial e aplique busca local nesta solução.
- Repita este procedimento por n vezes.
- Retorne a melhor solução encontrada.
- *Problema*: soluções aleatoriamente geradas em geral possuem baixa qualidade.

Multi-Start

Algoritmo 10.4 (Multi-Start)
Entrada Número de repetições n.

Saída Solução s.

```
Multi_Start(n) :=
{ mantm a melhor soluo s* }
repete n vezes
  gera soluo aleatria s
  s := BuscaLocal(s)
end repeat
return s*
```

Cobrimento de Vértices

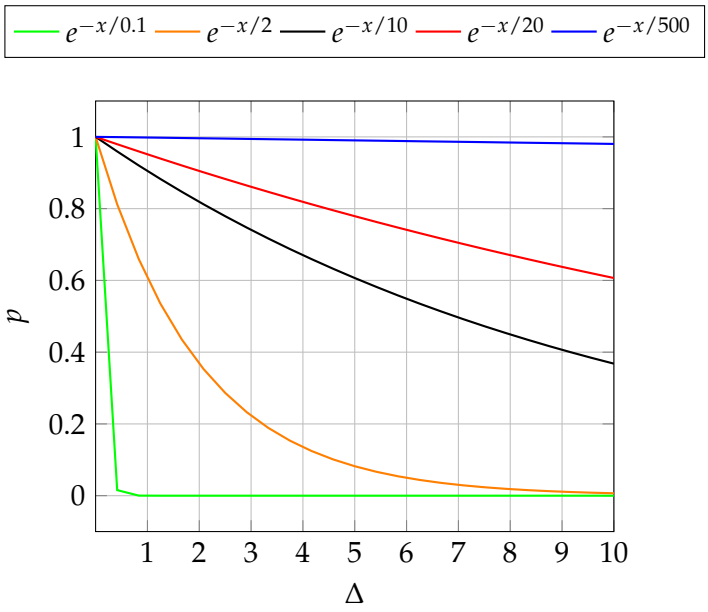
- Definição de vizinhança
- grafo sem vértices
- grafo estrela
- clique bipartido $K_{i,j}$
- grafo linha

10.2. Metropolis e Simulated Annealing

O algoritmo Metropolis

- Proposto por Metropolis et al. (1953).
- Simula o comportamento de um sistema físico de acordo com a mecânica estatística.
- Supõe temperatura constante
  - Um modelo básico define que a probabilidade de obter um sistema num estado com energia E é proporcional a $e^{-E/kT}$ (distribuição de Boltzmann), onde $T > 0$ é a temperatura, e $k > 0$ uma constante.
  - A função é monotônica decrescente em E: maior probabilidade de estar em um sistema de baixa energia.
  - Para T pequeno, a probabilidade de um sistema estar num estado de baixa energia é maior que ele estar num em estado de alta energia.
  - Para T grande, a probabilidade de passar para outra configuração qualquer do sistema é grande.

A distribuição de Boltzmann



Algoritmo Metropolis

- Estados do sistema são soluções candidatas.
- A energia do sistema é representada pelo custo da solução.
- Perturba a solução s gerando uma solução s'. Forma mais simples: seleciona um vizinho aleatório $s' \in \mathcal{N}(s)$.
- Se $E(s') \leq E(s)$ atualize a nova solução para s'.
- Caso contrário, $\Delta E = E(s') - E(s) > 0$.
- A solução s' passa ser a solução atual com probabilidade $e^{-\Delta E/kT}$.
- *Característica marcante*: permite movimentos de melhora e, com baixa probabilidade, também de piora.

Metropolis

Algoritmo 10.5 (Metropolis)
Entrada Uma solução inicial s e uma temperatura T.

Saída Solução s' com $c(s') \leq c(s)$.

```
Metropolis(s, T, k)=
  do
    seleciona s' ∈ N(s) aleatoriamente
    seja Δ := f(s') − f(s)
    if Δ ≤ 0 then
      atualiza s := s'
    else
      atualiza s := s' com probabilidade e^{−Δ/T}
    end if
  until critério de parada satisfeito
  return s
```

Observação 10.1
Para $T \to \infty$ o algoritmo executa um *passeio aleatório* no grafo das soluções com a vizinhança definida. Para $T \to 0$ o algoritmo se aproxima a uma busca local. ◊

Simulated Annealing

- Proposto por Cerny (1985) e Kirkpatrick et al. (1983).
- Simula um processo de recozimento.
- Recozimento: processo da física que aquece um material a uma temperatura bem alta e resfria aos poucos, dando tempo para o material alcançar seu estado de equilíbrio
- Recozimento simulado: parte de uma alta temperatura e baixa gradualmente. Para cada temperatura, permite um número máximo de saltos (dois laços encadeados)

Simulated Annealing

Algoritmo 10.6 (Simulated Annealing)
Entrada Solução inicial s, temperatura T, fator de esfriamento $r \in (0,1)$, número inteiro I.

Saída Solução s' tal que $f(s') \leq f(s)$.

```
SimulatedAnnealing(s, T, k, r, I) :=
  repeat até sistema ``esfriado''
    repeat I vezes
      seleciona s' ∈ N(s) aleatoriamente
      seja Δ := f(s') − f(s)
      if Δ ≤ 0 then
          s := s'
        else
          s := s' com probabilidade e^{−Δ/T}
        end fi
    end repeat
    T := rT
  end repeat
return s
```

Determinando uma temperatura inicial e final adequada é importante para não gastar tempo desnecessário com temperaturas em que o algoritmo se comporta como passeio aleatório ou busca local.

Exemplo 10.2 (Temperatura inicial)
Define uma probabilidade p_i. Executa uma versão rápida (I pequeno) do algoritmo para determinar uma temperatura inicial tal que um movimento é aceito com probabilidade p_i. ◊

Exemplo 10.3 (Temperatura final)
Define uma probabilidade p_f. Para cada nível de temperatura em que os movimentos foram aceitos com probabilidade menos que p_f incrementa um contador. Zera o contador caso uma nova melhor solução é encontrada. Caso o contador chega em 5, termina. ◊

10.3. GRASP

GRASP

Figura 10.3.: Mauricio G. C. Resende

- *GRASP*: greedy randomized adaptive search procedure
- Proposto por Mauricio Resende e Thomas Feo (1989).
- Mauricio Resende: Pesquisador da AT&T, Departamento de Algoritmos e Otimização

GRASP

- Método multi-start, em cada iteração
  1. Gera soluções com um procedimento guloso-randomizado.
  2. Otimiza as soluções geradas com busca local.

Algoritmo 10.7 (GRASP)
Entrada Parâmetro α.

Saída A melhor solução encontrada.

```
GRASP(α, ...)=
  { a busca mantém a melhor solução encontrada s* }
  do
    s := Guloso − Randomizado(α)
    s := BuscaLocal(s)
    atualiza s* caso f(s) < f(s*)
  until critério de parada satisfeito
  return s*
```

Construção gulosa-randomizada

- Motivação: Um algoritmo guloso gera boas soluções inicias.
- Problema: Um algoritmo determinístico produz sempre a mesma solução.
- Logo: Aplica um algoritmo guloso, que não escolhe *o melhor* elemento, mas escolhe randomicamente entre os α% *melhores* candidatos.
- O conjunto desses candidatos se chama *restricted candidate list* (RCL).

Construção gulosa-randomizada: Algoritmo guloso

```
Guloso() :=
  S := ()

  while S = (s_1,...,s_i) com i < n do
    entre todos candidatos C para s_{i+1}:
      escolhe o melhor s ∈ C
    S := (s_1,...,s_i,s)
  end while
```

Construção gulosa-randomizada: Algoritmo guloso

```
Guloso-Randomizado(α) :=
  S := ()

  while S = (s_1,...,s_i) com i < n do
    entre todos candidatos C para s_{i+1}:
      forma a RCL com os α\% melhores candidatos em C
      escolhe randomicamente um s ∈ RCL
    S := (s_1,...,s_i,s)
  end while
```

GRASP

Algoritmo 10.8 (GRASP)
Entrada Parâmetro α.

Saída Uma solução s^*.

```
GRASP(α)=
  do
    y := Guloso − Randomizado(α)
    y := BuscalLocal(y)
    atualiza a melhor solução s*
  until critério de parada satisfeito
  return s*
```

GRASP: Variações

- *long term memory*: hash table (para evitar otimizar soluções já vistas)
- Parâmetros: s_0, $\mathcal{N}(x)$, $\alpha \in [0,1]$ (para randomização), tamanho das listas (conj. elite, rcl, hash table), número de iterações,

GRASP com memória

- O GRASP original não havia mecanismo de memória de iterações passadas
- Atualmente toda implementação de GRASP usa conjunto de soluções elite e religação por caminhos (*path relinking*)
- *Conjunto de soluções elite*: conjunto de soluções diversas e de boa qualidade
  - uma solução somente é inserida se for melhor que a melhor do conjunto ou se for melhor que a pior do conjunto e diversa das demais
  - a solução a ser removida é a de pior qualidade
- *Religação por Caminhos*: a partir de uma solução inicial, modifique um elemento por vez até que se obtenha uma solução alvo (do conjunto elite)
- soluções intermediárias podem ser usadas como soluções de partida

Comparação entre as metaheurísticas apresentadas

- Metaheurísticas: Simulated annealing (SA), Multi-Start Search (MS), GRASP
- SA tem apenas um ponto de partida, enquanto que os outros dois métodos testa diversos
- SA permite movimento de piora, enquanto que os outros dois métodos não
- SA é baseado em um processo da natureza, enquanto que os outros dois não

10.4. Busca Tabu

Busca Tabu (*Tabu Search*)

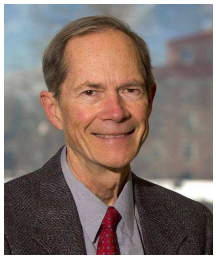


Figura 10.4.: Fred Glover (*1937)

- Proposto por Fred Glover em 1986 (princípios básicos do método foram propostos por Glover ainda em 1977)
- Professor da Universidade do Colorado, EUA

Busca Tabu (BT)

- Assim como em simulated annealing (SA) e VNS, TB é baseada inteiramente no processo de busca local, movendo-se sempre de uma solução s para uma solução s'
- Assim com em SA, também permite movimentos de piora
- Diferente de SA que permite movimento de piora por randomização, tal movimento na BT é determinístico
- A base do funcionamento de Busca Tabu é o uso de memória segundo algumas regras
- O nome *Tabu* tem origem na proibição de alguns movimentos durante a busca

Busca Tabu (BT)

- Mantém uma lista T de movimentos tabu
- A cada iteração se move para o melhor vizinho, desde que não faça movimentos tabus
- Permite piora da solução: o melhor vizinho pode ser pior que o vizinho atual!
- São inseridos na lista tabu elementos que provavelmente não direcionam a busca para o ótimo local desejado. Ex: último movimento executado
- o tamanho da lista tabu é um importante parâmetro do algoritmo
- Critérios de parada: quando todos movimentos são tabus ou se x movimentos foram feitos sem melhora

Busca Tabu: Conceitos Básicos e notação

- s: solução atual
- s^*: melhor solução
- f^*: valor de s*
- $\mathcal{N}(s)$: Vizinhança de s.
- $\tilde{\mathcal{N}}(s) \subset \mathcal{N}(s)$: possíveis (não tabu) soluções vizinhas a serem visitadas
- *Soluções*: inicial, atual e melhor
- *Movimentos*: atributos, valor
- *Vizinhança*: original, modificada (reduzida ou expandida)

Movimentos Tabu

- Um movimento é classificado como *tabu* ou *não tabu* pelas *regras de ativação tabu*
- em geral, as regras de ativação tabu classificam um movimento como tabu se o movimento foi recentemente realizado
- *Memória de curta duração (MCD)* - também chamada de *lista tabu*: usada para armazenar os movimentos tabu
- duração tabu (*tabu tenure*) é o número de iterações em que o movimento permanecerá tabu
- dependendo do tamanho da MCD um movimento pode deixar de ser tabu antes da duração tabu estabelecida
- A MCD em geral é implementada como uma lista circular
- O objetivo principal da MCD é evitar ciclagem e retorno a soluções já visitadas
- os movimentos tabu também colaboram para a busca se mover para outra parte do espaço de soluções, em direção a um outro mínimo local

Busca Tabu

Algoritmo 10.9 (BuscaTabu)
Entrada uma solução s

Saída uma solução $s' : f(s') \leq f(s)$

```
BuscaTabu(s)=
  { mantém a melhor solução s* }
  Inicialização:
    T := ∅
  while critério de parada não satisfeito
    s := seleciona s' ∈ Ñ(s) com min f(s')
    insira movimento em T (a lista tabu)
  end while
  return s*
```

Busca Tabu (BT)

- critérios de parada:
  - número de iterações (N_{max})
  - número interações sem melhora
  - quando s^* atinge um certo valor mínimo (máximo) estabelecido
- Um movimento não é executado se for tabu, ou seja, se possuir um ou mais atributos tabu-ativos
- Pode ser estabelecida uma regra de uso de um movimento tabu (critério de aspiração)
  - *Critério de aspiração por objetivo*: se o movimento gerar uma solução melhor que $s*$, permite uso do movimento tabu
  - *Critério de aspiração por direção*: o movimento tabu é liberado se for na direção da busca (de melhora ou piora)

Busca Tabu: mecanismos auxiliares

- *intensificação*: a idéia é gastar mais "esforço" em regiões do espaço de busca que parece mais promissores. Isso pode ser feito de diversas

maneiras (exemplo, guardar o número de interações com melhora consecutiva). Nem sempre este a intensificação traz benefícios.

- *Diversificação*: recursos algorítmicos que forçam a busca para um espaço de soluções ainda não explorados.
  - uso de memória de longo prazo (exemplo, número de vezes que a inserção de um elemento provocou melhora da solução)
  - Estratégia básica: forçar a inserção de alguns poucos movimentos pouco executados e reiniciar a busca daquele ponto
  - Estratégia usada para alguns problemas: permitir soluções infactíveis durante algumas interações

Busca Tabu: variações

- Várias listas tabus podem ser utilizadas (com tamanhos, duração, e regras diferentes)
- BT probabilístico: os movimentos são avaliados para um conjunto selecionado aleatoriamente $N'(s) \in \tilde{N}(s)$. Permite usar uma lista tabu menor, acontece menos ciclagem.
- A duração tabu pode variar durante a execução

Comparação entre as metaheurísticas apresentadas até então

- Metaheurísticas: Simulated annealing (SA), Multi-Start Search (MSS), GRASP, BT
- SA e BT têm apenas um ponto de partida, enquanto que os outros dois métodos testa diversos
- SA e BT permitem movimentos de piora, enquanto que os outros dois métodos não
- SA é baseado em um processo da natureza, enquanto que os outros métodos não

Parâmetros e decisões das metaheurísticas

- SA:
  - *Parâmetros*: temperatura inicial, critério de parada, variável de resfriamento
  - *Decisões*: vizinhança, solução inicial
- GRASP:
  - *Parâmetros*: s_0, $\mathbb{N}(x)$, $\alpha \in [0,1]$ (para randomização), tamanho das listas (conj. elite, rcl, hash table), critério de parada
  - *Decisões*: vizinhança, solução inicial (s_0), randomização da s_0, atualizações do conjunto elite
- BT:
  - *Parâmetros*: tamanho da lista tabu, critério de parada
  - *Decisões*: vizinhaça, critérios para classificar movimento tabu

10.5. Variable Neighborhood Search

Figura 10.5.: Pierre Hansen

Variable Neighborhood Search

- Proposto por Hansen e Mladenović (1997).
- Método que explora mais que uma vizinhança.
- Explora sistematicamente as seguintes propriedades:
  - O mínimo local de uma vizinhança não é necessariamente mínimo para outra vizinhança
  - Um mínimo global é um mínimo local com respeito a todas as vizinhanças
  - Para muitos problemas, os mínimos locais estão localizados relativamente próximos no espaço de busca para todas as vizinhanças

Os métodos usando k vizinhanças $\mathcal{N}_1, \ldots, \mathcal{N}_k$ sempre voltam a usar a primeira vizinhança, caso um movimento melhora a solução atual. Caso contrário eles passam para próxima vizinhança. Isso é o movimento básico:

Algoritmo 10.10 (Movimento)
Entrada Solução atual s, nova solução s', vizinhança atual k.

Saída Uma nova solução s e uma nova vizinhança k.

```
GRASP(α, ...)=
  { a busca mantém a melhor solução encontrada s* }
  do
    s := Guloso − Randomizado(α)
    s := BuscaLocal(s)
    atualiza s* caso f(s) < f(s*)
  until critério de parada satisfeito
  return s*
```

Com isso podemos definir uma estratégia simples, chamada *Variable Neighborhood Descent* (VND).

Algoritmo 10.11 (VND)
Entrada Solução inicial s, conjunto de vizinhanças $\mathcal{N}_i$, $i \in [m]$.

Saída Solução s.

```
VND(s,{N_i})=
  k := 1
  // até chegar num mínimo local
  // para todas vizinhanças
  while k ≤ m
    encontra o melhor vizinho s' ∈ N_k(s)
    (s,k) := Movimento(s,s',k)
  end while
  return s
```

Uma versão randomizada é o *reduced variable neighborhood search*.

Algoritmo 10.12 (rVNS)
Entrada Solução inicial s, conjunto de vizinhanças $\mathcal{N}_i$, $i \in [m]$.

Saída Solução s.

```
rVNS(s, {N_i}) =
  until critério de parada satisfeito
    k := 1
    while k ≤ m do
      seleciona vizinho aleatório s' ∈ N_k(s) { shake }
      (s, k) := Movimento(s, s', k)
    end while
  end until
  return s
```

Uma combinação do rVNS com uma busca local é o *Variable Neighborhood Search* (VNS) básico.

Algoritmo 10.13 (VNS)
Entrada Solução inicial s, um conjunto de vizinhanças $\mathcal{N}_i$, $i \in [m]$.

Saída Solução s.

```
VNS(s, {N_i}) =
  until critério de parada satisfeito
    k := 1
    while k ≤ m do
      seleciona vizinho aleatório s' ∈ N_k(s) { shake }
      s'' := BuscaLocal(s')
      (s, k) := Movimento(s, s'', k)
  end until
  return s
```

Observação 10.2
A busca local em VNS pode usar uma vizinhança diferente das vizinhanças que perturbam a solução atual. Também é possível usar o VND no lugar da busca local. ◊

10.6. Algoritmo Guloso Iterado

Algoritmos de construção repetida independente como GRASP e Multi-Start criam diversas soluções durante a execução, mas não utilizam a informação obtida por iterações anteriores para ajudar na composição de novas soluções. O algoritmo guloso iterado proposto por Ruiz e Stützle (2007) utiliza parte

da solução encontrada anteriormente para tentar achar uma nova solução melhor.
O algoritmo guloso iterado cria uma solução inicial e iterativamente destrói e reconstrói soluções de forma a gerar soluções novas. A cada etapa parte da solução é removida. tornando a solução parcial, então o algoritmo gera uma nova solução completa de forma gulosa à partir dessa solução parcial. Uma vez gerada a solução nova verificamos se a solução será aceita ou descartada. Caso ela seja melhor que a solução atual ela é aceita, caso seja pior é aceita com chance dada pela perda de qualidade utilizando o critério de Metropolis. O pseudo-código está no Algoritmo 10.14.

Algoritmo 10.14 (Busca Gulosa Iterada)
Entrada: Número de repetições n, temperatura T, uma solução inicial s. **Saída**: Melhor solução encontrada s^*.

```
IG(s):=
  { manter melhor solução s* }
  for n vezes
    s' = s
    Destrói parte de s'
    Reconstrói s' gulosamente.
    Δ = f(s') − f(s)
    s = s' com probabilidade min{1, e^{−Δ/T}}
  end for
  return s*
```

No algoritmo utilizamos um número fixo de iterações mas podemos utilizar a qualidade da solução ou o tempo de execução como critério de parada. Note que utilizamos o a mesma estratégia que o algoritmo de Metropolis para permitir soluções a transição para soluções qualidade pior que a anterior, entretanto não utilizamos resfriamento (como utilizado na Têmpera Simulada). A destruição e reconstrução em sequencia podem ser consideradas uma perturbação da solução atual, pois podemos ter uma solução nova de qualidade melhor ou pior, portanto pode ser útil colocar algum método de melhoria, como uma busca local, após a reconstrução.
No caso do caixeiro viajante podemos fazer a destruição removendo um número constante de arestas aleatórias do ciclo hamiltoniano, e a reconstrução com a heurística do vizinho mais próximo. No caso da max-SAT podemos

tornar alguns bits aleatórios não definidos para destruir parte da solução, então construímos uma nova solução completa re-definindo estes bit em (ordem aleatória), cada vez maximizando o número de cláusulas satisfeitas.

11. Heurísticas inspirados da natureza

11.1. Algoritmos Genéticos e meméticos

Algoritmos Genéticos

- Proposto na década de 60 por Henry Holland.
- Professor da Faculdade de Engenharia Elétrica e de Computação da Universidade de Michigan/EUA.
- Seu livro: Adaptation in Natural and Artificial Systems (1975).

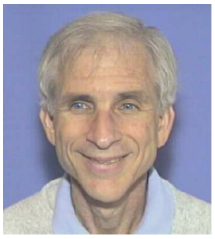


Figura 11.1.: John Henry Holland (*1929,+2015)

Algoritmos genéticos

- Foi proposto com o objetivo de projetar software de sistemas artificiais que reproduzem processos naturais.
- Baseados na evolução natural das espécies.
- Por Darwin: indivíduos mais aptos têm mais chances de perpetuar a espécie.
- Mantém uma população de soluções e não uma única solução por vez.
- Usa regras de transição probabilísticas, e não determinísticas.
- Procedimentos: avaliação, seleção, geração de novos indivíduos (recombinação), mutação.
- Parada: número x de gerações total, número y de gerações sem melhora.

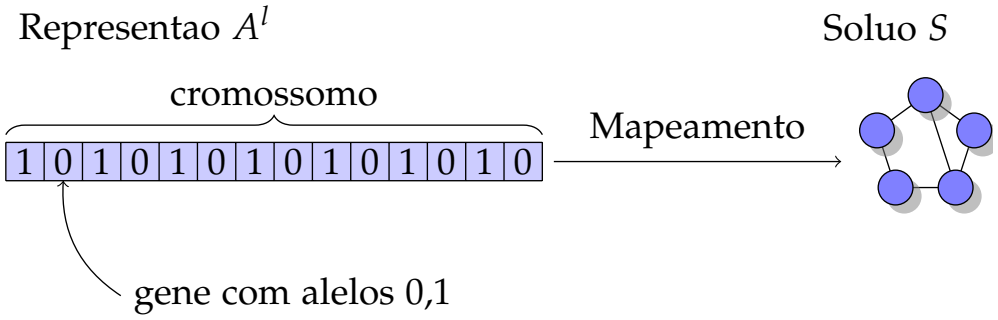
Algoritmos genéticos: Características

- Varias soluções (“população”).
- Operações novas: Recombinação e mutação.
- Separação da representação (“genótipo”) e formulação “natural” (fenótipo).

Algoritmos Genéticos: Noções

- *Genes*: Representação de um elemento (binário, inteiro, real, arco, etc) que determine uma característica da solução.
- *Alelo*: Instância de uma gene.
- *Cromossomo*: Uma string de genes que compõem uma solução.
- *Genótipo*: Representação genética da solução (cromossomos).
- *Fenótipo*: Representação "física" da solução.
- *População*: Conjunto de cromossomos.

Algorítmos genéticos: Representação e Solução



Algoritmos Genéticos: exemplos

- Problema de partição de conjuntos

  Alelos: 0 ou 1

  Cromossomo: 00011010101011110110

- Problema do Caixeiro viajante

  Alelos: valores inteiros entre 1 e n

  Cromossomo: 1 5 3 6 8 2 4 7

Procedimentos dos Algoritmos Genéticos

- *Codificação*: genes e cromossomos.
- *Initialização*: geração da população inicial.
- *Função de Avaliação (fitness)*: função que avalia a qualidade de uma solução.
- *Seleção de pais*: seleção dos indivíduos para crossover.
- *Operadores genéticos*: crossover, mutação
- *Parâmetros*: tamanho da população, percentagem de mutação, critério de parada

Algoritmos Genéticos

Algoritmo 11.1 (AlgoritmoGenético)
Entrada Parâmetros do algoritmo.

Saída Melhor solução encontrada para o problema.

```
Inicialização e avaliação inicial
while (critério de parada não satisfeito) do
  repeat
    if (critério para recombinação) then
      selecione pais
      recombina e gera um filho
    end if
    if (critério para mutação) then
      aplica mutação
    end if
  until (descendentes suficientes)
  selecione nova população
end while
```

População Inicial: geração

- Soluções aleatórias.

- Método construtivo (ex: vizinho mais próximo com diferentes cidades de partida).
- Heurística construtiva com perturbações da solução.
- Pode ser uma mistura das opções acima.

População inicial: tamanho

- População maior: Custo alto por iteração.
- Populaçao menor: Cobertura baixa do espaço de busca.
- Critério de Reeves: Para alfabeto binário, população randômica:
  Cada ponto do espaço de busca deve ser alcancável através de recombinações.
- Consequencia: Probabilidade que cada alelo é presente no gene i: $1 - 2^{1-n}$.
- Probabilidade que alelo é presente em todos gene: $(1 - 2^{1-n})^l$.
- Exemplo: Com $l = 50$, para garantir cobertura com probabilidade 0.999:
$$n \geq 1 - \log_2 \left(1 - \sqrt[50]{0.999}\right) \approx 16.61$$

Terminação

- Tempo.
- Número de avaliações.
- Diversidade. Exemplo: Cada gene é dominado por um alelo, i.e. 90% dos indivíduos tem o mesmo alelo.

Próxima Geração

- Gerada por recombinação e mutação (soluções aleatórias ou da população anterior podem fazer parte da próxima geração).
- Estratégias:

  - Recombinação *e* mutação.
  - Recombinação *ou* mutação.
- Regras podem ser randomizadas.
- Exemplo: Taxa de recombinação e taxa de mutação.
- Exemplo: Número de genes mutados.

Mutação

- *Objetivo*: Introduzir elementos diversificados na população e com isso possibilitar a exploração de uma outra parte do espaçõ de busca.
- Exemplo para representação binária: flip de k bits.
- Exemplo para o PCV: troca de posição entre duas cidades.

Recombinação

- Recombinação (ingl. crossover): combinar características de duas soluções para prover uma nova solução potencialmente com melhor fitness.
- Explora o espaço entre soluções.
- Crossover clássicos: one-point recombinação e two-points recombinação.

One-point crossover

Escolha um número aleatório k entre 1 e n. Gere um filho com os primeiros k bits do pai A e com os últimos $n - k$ bits do pai B

- *Problema de particação*: aplicação direta do conceito
- *Problema do Caixeiro Viajante*: copie os primeiros k elementos do pai A e as demais $n - k$ posições preenche com as cidades faltantes, segundo a ordem em que elas aparecem no pai B

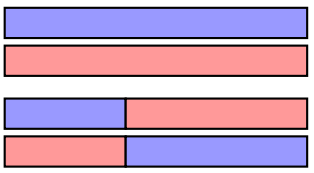


Figura 11.2.: Recombinação de um ponto.

Recombinação de dois pontos

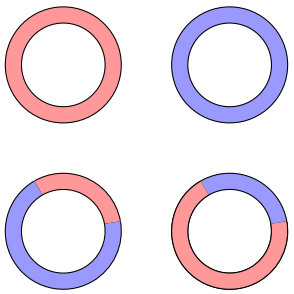


Figura 11.3.: Recombinação de dois pontos.

Exemplo: Strategic Arc Crossover

- Selecione todos os pedaços de rotas (string) com 2 ou mais cidades que são iguais nas duas soluções
- Forme uma rota através do algoritmo de vizinho mais próximo entre os pontos extremos dos strings

Recombinação: Seleção dos pais

- A probabilidade de uma solução ser pai num processo de crossover deve depender do seu fitness.
- Variações:
  - Probabilidade proporcional com fitness.
  - Probabilidade proporcional com ordem.

Estratégia adotada pelos operadores

Inúmeros operadores podem ser propostos para cada problema. O ideal é combinar características do operador usado, com outros operadores (mutação, busca local) usados no GA. Basicamente um crossover é projetado da seguinte forma:

- Encontre similaridades entre A e B e insira $S = A \cap B$ no filho.
- Defina conjuntos S_{in} e S_{out} de características desejáveis e não desejáveis.
- Projete um operador que mantenha ao máximo elementos de S e S_{in}, minimizando o uso de elementos de S_{out}.

Nova População

- Todos os elementos podem ser novos.
- Alguns elementos podem ser herdados da população anterior.
- Elementos novos podem ser gerados.
- Exemplos, com população de tamanho λ que gera μ filhos.
  (λ, μ) Seleciona os λ melhores dos filhos.
  $(\lambda + \mu)$ Seleciona os λ melhores em toda população.

Estrutura da População

Em geral, população estruturada garante melhores resultados. A estrutura da população permite selecionar pais para crossover de forma mais criteriosa. Algumas estruturas conhecidas

- *Divisão em Castas*: 3 partições A, B e C (com tamanhos diferentes), sendo que os melhores indivíduos estão em A e os piores em C.
- *Ilhas*: a população é particionada em subpopulações que evoluem em separado, mas trocam indivíduos a cada período de número de gerações.
- *População organizada como uma árvore*.

Exemplo: População em castas

- Recombinação: Somente entre indivíduos da casta A e B ou C para manter diversidade.
- Nova população: Manter casta "elite" A, re-popular casta B com filhos, substituir casta C com soluções randômicas.

Exemplo: População em árvore

- Considere uma árvore ternária completa, em que cada nó possui duas soluções (pocket e current).
- A solução current é a solução atual armazenada naquela posição da árvore.
- A solução pocket é a melhor já tida naquela posição desde a primeira geração.
- A cada solução aplique *exchange* (se a solução current for melhor que a pocket, troque-as de posição)
- Se a solução pocket de um filho for melhor que a do seu pai, troque o nó de posição.

Algoritmos Meméticos

- Proposto por Pablo Moscato, Newcastle, Austrália.
- Ideía: Informação "cultural" pode ser adicionada a um indivíduo, gerando um algoritmo memético.
- *Meme*: unidade de informação cultural.

Figura 11.4.: Pablo Moscato

Algoritmos Meméticos

- Um procedimento de busca local pode inserir informação de boa qualidade, e não genética (memes).
- Faz uso de um procedimento de busca local (em geral aplicado à solução gerada pelo procedimento de recombinação).
- Geralmente trabalha com populações menores.

Comparação entre as Metaheurísticas Apresentadas

- Quais que dependem de randomização? SA, GRASP, GA
- Quais que geram apenas uma solução inicial em todo processo? BT, SA
- Quais mantêm um conjunto de soluções, em vez de considerar apenas uma? GA
- Quais são inspiradas em processos da natureza? GA, BT
- Qual gera os melhores resultados?

Existem outras Metaheurísticas

Handbook of Metaheuristics, por Fred W. Glover (Editor), Gary A. Kochenberger (Editor) Kluwer 2002.

Considerações Finais

- O desempenho de uma metaheurística depende muito de cada implementação
- As metaheurísticas podem ser usadas de forma hibridizada
- Técnicas de otimização multiobjetivo tratam os casos de problemas com mais de um objetivo (Curva de Pareto)

Exercício

- Problema de alocação: atender n clientes por m postos de atendimento (um posto é instalado no local onde se encontra um cliente)
- Entrada: distâncias entre cada par de clientes
- Problema: Determinar em que locais instalar os postos, de forma a minimizar a soma das distâncias de cada cliente a um ponto de atendimento
- Propor uma heurística construtiva e uma busca local.

Comparação entre as Metaheurísticas

- Quais que permitem movimento de piora? BT, SA
- Quais que não dependem de randomização? BT
- Quais que geram apenas uma solução inicial em todo processo? BT, SA
- Quais mantêm um conjunto de soluções, em vez de considerar apenas uma?
- Qual gera os melhores resultados?

Parte IV.

Appéndice

A. Conceitos matemáticos

$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ e $\mathbb{R}$ denotam os conjuntos dos números naturais sem 0, inteiros, racionais e reais, respectivamente. Escrevemos também $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, para qualquer conjunto C, $C_+ = \{x \in C | x \geq 0\}$ e $C_- = \{x \in C \mid x \leq 0\}$. Por exemplo

$$\mathbb{R}_+ = \{x \in \mathbb{R} \mid x \geq 0\}.^1$$

Para um conjunto finito S, $\mathcal{P}(S)$ denota o conjunto de todos subconjuntos de S.
Denotamos por $A = (a_{ij}) \in F^{m \times n}$ uma *matriz* de m linhas e n colunas com elementos em F. A i-ésima linha é a_i, com $a_i^t \in F^n$ e a j-ésima coluna de A é $a^j \in F^m$.

Definição A.1 (Pisos e tetos)
Para $x \in \mathbb{R}$ o *piso* $\lfloor x \rfloor$ é o maior número inteiro menor que x e o *teto* $\lceil x \rceil$ é o menor número inteiro maior que x. Formalmente

$$\lfloor x \rfloor = \max\{y \in \mathbb{Z} \mid y \leq x\}$$
$$\lceil x \rceil = \min\{y \in \mathbb{Z} \mid y \geq x\}$$

O *parte fracionário* de x é $\{x\} = x - \lfloor x \rfloor$.

Observe que o parte fracionário sempre é positivo, por exemplo $\{-0.3\} = 0.7$.

Proposição A.1 (Regras para pisos e tetos)
Pisos e tetos satisfazem

$$x \leq \lceil x \rceil < x + 1 \tag{A.1}$$
$$x - 1 < \lfloor x \rfloor \leq x \tag{A.2}$$

[1] Alguns autores usam $\mathbb{R}^+$.

B. Formatos

Este capítulo contém um breve resumo dos formatos CPLEX lp, Julia/JuMP e AMPL/MathProg usados para especificar problemas de otimização linear. CPLEX LP é um formato simples, AMPL[1] é uma linguagem completa para definir problemas de otimização, com elementos de programação, comandos interativos e um interface para diferentes resolvedores de problemas. Por isso CPLEX LP serve para modelos pequenos. Aprender AMPL precisa mais investimento, que rende em aplicações maiores. AMPL tem o apoio da maioria das ferramentas disponíveis.
Vários outros formatos estão em uso, a maioria deles comerciais. Exemplos são ZIMPL, GAMS, LINGO, e MPS (Mathematical programming system).

B.1. CPLEX LP

Uma gramática simplificada[2] do formato CPLEX LP é

⟨*specification*⟩ ::= ⟨*objective*⟩
⟨*restrictions*⟩?
⟨*bounds*⟩
⟨*general*⟩?
⟨*binary*⟩?
'End'

⟨*objective*⟩ ::= ⟨*goal*⟩ ⟨*name*⟩? ⟨*linear expression*⟩

⟨*goal*⟩ ::= 'MINIMIZE' | 'MAXIMIZE' | 'MIN' | 'MAX'

⟨*restrictions*⟩ ::= 'SUBJECT TO' ⟨*restriction*⟩+

⟨*restriction*⟩ ::= ⟨*name*⟩? ⟨*linear expression*⟩ ⟨*cmp*⟩ ⟨*number*⟩

[1] A sigla AMPL significa "A mathematical programming language". O nome também sugere uma funcionalidade "ampla" ("ample" em inglês).
[2] A gramática não contém as especificações "semi-continuous" e "SOS".

⟨*cmp*⟩ ::= '<' | '<=' | '=' | '>' | '>='

⟨*linear expression*⟩ ::= ⟨*number*⟩ ⟨*variable*⟩ (('+' | '-') ⟨*number*⟩ ⟨*variable*⟩)*

⟨*bounds*⟩ ::= 'BOUNDS' ⟨*bound*⟩+

⟨*bound*⟩ ::= ⟨*name*⟩? (⟨*limit*⟩ '<=' ⟨*variable*⟩ '<=' ⟨*limit*⟩
| ⟨*limit*⟩ '<=' ⟨*variable*⟩
| ⟨*variable*⟩ '<=' ⟨*limit*⟩
| ⟨*variable*⟩ '=' ⟨*number*⟩
| ⟨*variable*⟩ 'free')

⟨*limit*⟩ ::= 'infinity' | '-infinity' | ⟨*number*⟩

⟨*general*⟩ ::= 'GENERAL' ⟨*variable*⟩+

⟨*binary*⟩ ::= 'BINARY' ⟨*variable*⟩+

Todas variáveis x tem a restrição padrão $0 \leq x \leq +\infty$. Caso outros limites são necessárias, eles devem ser informados na seção "BOUNDS". As seções "GENERAL" e "BINARY" permitem restringir variáveis para $\mathbb{Z}$ e $\mathbb{B}$, respectivamente.
As palavras-chaves também podem ser escritas com letras minúsculas: o formato permite algumas abreviações não listadas acima (por exemplo, escrever "s.t" ou "st" ao invés de "subject to").
Um comentário até o final da linha inicia com "\". Uma alternativa são comentários entre "*" e "*\".

Exemplo B.1 (Problema (1.1) no formato CPLEX LP)

```
Maximize
        lucro:  0.2 c + 0.5 s

Subject To
        ovo:    c + 1.5 s   <= 150   \ um comentrio
        acucar: 50 c + 50 s <= 6000
        client1:c           <= 80
        client2:s           <= 60

Bounds
```

```
        0 <= c
        0 <= s
End
```

◊

Exemplo B.2
Problema de mochila 0-1 com 11 itens em formato CPLEX LP.

```
max 19x1+87x2+97x3+22x4+47x5+22x6+30x7+5x8+32x9+54x10+75x11
s.t
1x1+96x2+67x3+90x4+13x5+74x6+22x7+86x8+23x9+63x10+89x11<= 624
binary x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11
end
```

◊

Observação B.1
CPLEX LP permite constantes como `0.5e6` que representa 0.5×10^6. Outra interpretação dessa expressão é 0.5 vezes a variável e_6. Para evitar essa ambiguidade, variáveis não podem começar com a letra `e`. ◊

B.2. Julia/JuMP

Julia é uma linguagem para programação científica e JuMP (Julia for Mathematical Programming) uma biblioteca que permite embutir programas matemáticos diretamente em código Julia. Isso tem a vantagem de poder ler e processar os dados antes da solução, resolver, e continuar trabalhar com o resultado no mesmo programa.

Exemplo B.3 (Problema (1.1) em Julia/JuMP)

```
#!/usr/bin/env julia

using JuMP
using GLPKMathProgInterface

m = Model(solver=GLPKSolverMIP())

@variable(m, c)
@variable(m, s)

@objective(m, Max, 0.2*c+0.5*s)
```

```
@constraint(m, c    + 1.5*s <=  150)
@constraint(m, 50*c +  50*s <= 6000)
@constraint(m,    c         <=   80)
@constraint(m,            s <=   60)

status = solve(m)

if status == :Optimal
    println("A soluo tima  c=$(getvalue(c)) e s=$(getvalue(s))
     ↪  de valor $(getobjectivevalue(m)).")
end
```

◊

Diferente do CPLEX lp, Julia/JuMP permite expressar um único modelo para um problema e resolver para diferentes instâncias.

Exemplo B.4 (Exemplo (1.3) em Julia/JuMP)

```
#!/usr/bin/env julia

using JuMP
using GLPKMathProgInterface

n = 3
m = 3
a = [5,7,3]
b = [7,3,5]
c = [[3,4,100] [1,2,3] [100,4,3]]

mm = Model(solver=GLPKSolverMIP())

@variable(mm, x[1:m,1:n] >= 0)

@objective(mm, Min, sum(c[i,j]*x[i,j] for i=1:m, j=1:n))

@constraint(mm, [i=1:m], sum(x[i,j] for j=1:n) <= a[i])
@constraint(mm, [j=1:n], sum(x[i,j] for i=1:m) == b[j])
```

```
status = solve(mm)

if status == :Optimal
    println("A soluo tima  x=$(getvalue(x)) de valor
     ↪  $(getobjectivevalue(mm)).")
end
```

◊

B.3. AMPL

Objetos de modelagem

- Um modelo em AMPL consiste em
  - parâmetros,
  - variáveis,
  - restrições, e
  - objetivos
- AMPL usa *conjuntos* (ou arrays de múltiplas dimensões)

$$A : I \to D$$

que mapeiam um conjunto de índices $I = I_1 \times \cdots \times I_n$ para valores D.

Formato

- Parte do modelo

```
s1
...
sn
end;
```

com s_i sendo um comando ou uma declaração.

- Parte de dados

```
data
d1
...
dn
end;
```

com d_i sendo uma especificação de dados.

Tipo de dados

- Números: `2.0,-4`
- Strings: `'Comida'`
- Conjuntos: `{2,3,4}`

Expressões numéricas

- Operações básicas: `+,-,*,/,div,mod,less,**`

  Exemplo: `x less y`

- Funções: `abs,ceil,floor,exp`

  Exemplo: `abs(-3)`

- Condicional: `if x>y then x else y`

Expressões sobre strings

- AMPL converte números automaticamente em strings
- Concatenação de strings: `&`

  Exemplo: `x & ' unidades'`

Expressões para conjuntos de índices

- Uma dimensão
  - `t in S`: variável "dummy" t, conjunto S
  - `(t1,...tn) in S`: para conjuntos de tuplos
  - `S`: sem nomear a variável
- Multiplas dimensões
  - {`e1,...,en`} com e_i uma dimensão (acima).
- Variáveis "dummy" servem para referenciar e modificar.

  Exemplo: `(i-1) in S`

Conjuntos

- Conjunto básico: {`v1,...,vn`}
- Valores: Considerados como conjuntos com conjunto de índices de dimensão 0
- Índices: `[i1,...,in]`
- Sequências: `n1 ... n2 by d` ou `n1 ... n2`
- Construção: `setof I e`: $\{e(i_1, \ldots, i_n) \mid (i_1, \ldots, i_n) \in I\}$

  Exemplo: `setof {j in A} abs(j)`

Operações de conjuntos

- `X union Y`: União $X \cup Y$
- `X diff Y`: Diferença $X \setminus Y$
- `X symdiff Y`: Diferença simétrica $(X \setminus Y) \cup (Y \setminus X)$
- `X inter Y`: Intersecção $X \cap Y$
- `X cross Y`: Produto cartesiano $X \times Y$

Expressões lógicas

- Interpretação de números: n vale "v", sse $n \neq 0$.
- Comparações simples: `<`,`<=`,`=` ou `==`,`>=`,`>`,`<>` ou `!=`
- Pertinência: `x in Y`, `x not in Y`, `x !in Y`
- Subconjunto: `X within Y`, `X !within Y`, `X not within Y`
- Operadores lógicos: `&&` ou `and`, `||` ou `or`, `!` ou `not`
- Quantificação: com índices I, expressão booleana b

  `forall I b`: $\bigwedge_{(i_1,\ldots,i_n)\in I} b(i_1,\ldots,i_n)$

  `exists I b` $\bigvee_{(i_1,\ldots,i_n)\in I} b(i_1,\ldots,i_n)$

Declarações: Conjuntos

```
set N I [dimen n] [within S] [default e1] [:= e2]

param N I [in S] [<=,>=,!=,... n] [default e1] [:= e2]
```

- Nome N
- Conjunto de índices I (opcional)
- Conjunto de valores S
- Valor default e_1
- Valor inicial e_2

Declarações: Restrições e objetivos

```
subject to N I : e1 = e2 | e1 <= e2, e1 >= e2

minimize [I] : e

maximize [I] : e
```

Comandos

- `solve`: Resolve o sistema.
- `check [I] : b`: Valida expressão booleana b, erro caso falso.
- `display [I] : e1,...en`: Imprime expressões $e_1, \ldots, e_n$.
- `printf [I] : fmt,e1,...,en`: Imprime expressões $e-1, \ldots, e_n$ usando formato fmt.
- `for I : c`, `for I : {c1 ... cn}`: Laços.

Dados: Conjuntos

```
set N r1,...rn
```

Com nome N e records $r_1, \ldots, r_n$, cada record

- um tuplo: $v_1, \ldots, v_n$
  Exemplo: 1 2, 1 3, 2 2, 2 7
- a definição de uma fatia $(v_1|*, v_2|*, \ldots, v_n|*)$: depois basta de listar os elementos com $*$.
  Exemplo: (1 *) 2 3, (2 *) 2 7
- uma matriz

  ```
    : c1 c2  ... cn :=
  r1 a11 a12 ... a1n
  r2 a21 a22 ... a2n
         ...
  rm am1 am2 ... amn
  ```

  com `aij` "+"/"-" para inclusão/exclusão do par "`ri cj`" do conjunto.

Dados: Parâmetros

```
param N r1,...rn
```

Com nome N e records $r_1, \ldots, r_n$, cada record

- um valor $i_1, \ldots, i_n, v$

- a definição de uma fatia $[i_1|*, i_2|*, \ldots, i_n|*)$: depois basta definir índices com $*$.
- uma matriz

```
  : c1 c2  ... cn :=
r1 a11 a12 ... a1n
r2 a21 a22 ... a2n
       ...
rm am1 am2 ... amn
```

  com `aij` o valor do par "`ri cj`".

- uma tabela

```
param default v : s : p1 p2 ... pk :=
t11 t12 ... t1n  a11 a12 ... a1k
t21 t22 ... t2n  a21 a22 ... a2k
        ...
tm1 tm2     tmn  am1 am2 ... amk
```

  para definir simultaneamente o conjunto

```
  set s := (t11 t12 ... t1n), ... , (tm1 tm2 ... tmn);
```

  e os parâmetros

```
  param p1 default v := [t11 t12 ... t1n] a11, ..., [tm1
↪  tm2 ... tmn] am1;
  param p2 default v := [t11 t12 ... t1n] a12, ..., [tm1
↪  tm2 ... tmn] am2;
  ...
  param pk default v := [t11 t12 ... t1n] a1k, ..., [tm1
↪  tm2 ... tmn] amk;
```

Exemplo B.5 (Exemplo (1.1) em AMPL)

```
var c;  # nmero de croissants
var s;  # nmero de strudels
param lucro_croissant;     # o lucro por croissant
param lucro_strudel;   # o lucro por strudel
maximize lucro: lucro_croissant*c+lucro_strudel*s;
subject to ovo: c+1.5*s <= 150;
subject to acucar: 50*c+50*s <= 6000:
subject to croissant: c <= 80;
subject to strudel: s <= 60;
```

◊

Exemplo B.6 (Exemplo (1.3) em AMPL)

```
param n; # nmero de clientes
param m; # nmero de fornecedores
param a { 1..m }; # estoque
param b { 1..n }; # demanda
param c { 1..m, 1..n }; # custo transporte
var x { 1..m,1..n } >= 0;

minimize custo:
        sum { i in 1..m, j in 1..n } c[i,j]*x[i,j];
subject to limiteF { i in 1..m }:
        sum { j in 1..n } x[i,j] <= a[i];
subject to limiteC { j in 1..n }:
        sum { i in 1..m } x[i,j]  = b[j];

data;
param n := 3;
param m := 3;
param a := 1 5, 2 7, 3 3;
param b := 1 7, 2 3, 3 5;
param c :   1   2   3 :=
1           3   1 100
2           4   2   4
3         100   3   3;
end;
```

◊

C. Soluções dos exercícios

Solução do exercício 1.3.

$$\begin{aligned} \textbf{maximiza} \quad & 2A + B \\ \textbf{sujeito a} \quad & A \leq 6000, \\ & B \leq 7000, \\ & A + B \leq 10000, \\ & A, B \geq 0. \end{aligned}$$

Resposta: $A = 6000$, $B = 4000$, e $Z = 16000$.

Solução do exercício 1.4.
São necessárias cinco variáveis:

- x_1: número de pratos de lasanha comidos por Marcio
- x_2: número de pratos de sopa comidos por Marcio
- x_3: número de pratos de hambúrgueres comidos por Renato
- x_4: número de pratos de massa comidos por vini
- x_5: números de pratos de sopa comidos por vini

Formulação:

$$\begin{aligned} \textbf{maximiza} \quad & x_1 + x_2 + x_3 + x_4 + x_5 \\ \textbf{sujeito a} \quad & 4 \geq x_1 + x_2 \geq 2, \\ & 5 \geq x_3 \geq 2, \\ & 4 \geq x_4 + x_5 \geq 2, \\ & 70(x_2 + x_5) + 200x_1 + 100x_3 + 30x_4 \leq 1000, \\ & 30(x_2 + x_5) + 100x_1 + 100x_3 + 100x_4 \leq 800. \end{aligned}$$

Solução do exercício 1.5.
Sejam $l_1 \in \mathbb{R}$ e $l_2 \in \mathbb{R}$ o número de lampadas produzidas do tipo 1 e 2, respectivamente. Com isso podemos formular

$$\begin{aligned} \textbf{maximiza} \quad & l_1 + 2l_2 \\ \textbf{sujeito a} \quad & l_2 \leq 60, \\ & l_1 + 3l_2 \leq 200, \\ & 2l_1 + 2l_2 \leq 300, \\ & l_1, l_2 \geq 0. \end{aligned}$$

Solução do exercício 1.6.
Sejam $m \in \mathbb{R}$ e $a \in \mathbb{R}$ o número de janelas de madeira e de alumínio, respectivamente. Com isso podemos formular

$$\begin{aligned} \textbf{maximiza} \quad & 60m + 30a, \\ \textbf{sujeito a} \quad & m \leq 6, \\ & a \leq 4, \\ & 6m + 8a \leq 48, \\ & m, a \geq 0. \end{aligned}$$

Solução do exercício 1.8.
Com marcas J, O, M (Johnny Ballantine, Old Gargantua, Misty Deluxe) e misturas A, B, C temos as variáveis

$$x_{J,A}, x_{J,B}, x_{J,C}, x_{O,A}, x_{O,B}, x_{O,C}, x_{M,A}, x_{M,B}, x_{M,C}$$

que denotam o número de garrafas usadas por mistura.
Vamos introduzir ainda as variáveis auxiliares para o número de garrafas usadas de cada marca

$$x_J = x_{J,A} + x_{J,B} + x_{J,C}, \quad x_O = x_{O,A} + x_{O,B} + x_{O,C}, \quad x_M = x_{M,A} + x_{M,B} + x_{M,C}$$

e variáveis auxiliares para o número de garrafas produzidas de cada mistura

$$x_A = x_{J,A} + x_{O,A} + x_{M,A}, \quad x_B = x_{J,B} + x_{O,B} + x_{M,B}, \quad x_C = x_{J,C} + x_{O,C} + x_{M,C}.$$

Queremos maximizar o lucro em reais

$$68x_A + 57x_B + 45x_C - (70x_J + 50x_O + 40x_M)$$

respeitando os limites de importação

$$x_J \leq 2000, \quad x_O \leq 2500, \quad x_M \leq 1200$$

e os limites de percentagem

$$\begin{aligned} &x_{J,A} \geq 0.6x_A, \quad x_{M,A} \leq 0.2x_A, \\ &x_{J,B} \geq 0.15x_B, \quad x_{M,B} \leq 0.6x_B, \\ &x_{M,C} \leq 0.5x_C. \end{aligned}$$

Portanto, o sistema final é

$$\begin{aligned} \textbf{maximiza} \quad & 68x_A + 57x_B + 45x_C \\ & \quad - (70x_J + 50x_O + 40x_M), \\ \textbf{sujeito a} \quad & cx_J \leq 2000, \\ & x_O \leq 2500, \\ & x_M \leq 1200, \\ & x_{J,A} \geq 0.6x_A, \\ & x_{M,A} \leq 0.2x_A, \\ & x_{J,B} \geq 0.15x_B, \\ & x_{M,B} \leq 0.6x_B, \\ & x_{M,C} \leq 0.5x_C, \\ & x_m = x_{m,A} + x_{m,B} + x_{m,C}, && m \in \{J, O, M\}, \\ & x_m = x_{J,m} + x_{O,m} + x_{M,m}, && m \in \{A, B, C\}, \\ & x_{m,n} \geq 0, && m \in \{J, O, M\}, n \in \{A, B, C\}. \end{aligned}$$

Sem considerar a integralidade a solução ótima é produzir 2544.44 garrafas da mistura A, 3155.56 garrafas da mistura B e 0 garrafas da mistura C, com as percentagens

- A: 60% Johnny Ballantine, 20% Old Gargantua, 20% Misty Deluxe
- B: 15% Johnny Ballantine, 63% Old Gargantua, 22% Misty Deluxe

Solução do exercício 1.9.

Com t_1 o número de TVs de 29" e t_2 de 31" temos

$$\begin{aligned} \textbf{maximiza} \quad & 120t_1 + 80t_2, \\ \textbf{sujeito a} \quad & t_1 \leq 40, \\ & t_2 \leq 10, \\ & 20t_1 + 10t_2 \leq 500, \\ & t_1, t_2 \geq 0. \end{aligned}$$

Solução do exercício 1.10.
Sejam $V = \{V_1, V_2\}$ e $NV = \{NV_1, NV_2, NV_3\}$ os conjuntos de óleos vegetais e não vegetais e $O = V \cup NV$ o conjunto do todos óleos. Seja ainda c_i o custo por tonelada do óleo $i \in O$ e a_i a acidez do óleo $i \in O$. (Por exemplo $c_{V_1} = 110$ e $a_{NV_2} = 4.2$.) Com variáveis x_i (toneladas refinadas do óleo $i \in O$) e x_o (quantidade total de óleo produzido) podemos formular

$$\begin{aligned} \textbf{maximiza} \quad & 150x_o - \sum_{i \in O} c_i x_i & & \\ \textbf{sujeito a} \quad & \sum_{i \in V} x_i \leq 200, & & \text{limite óleos vegetais,} \\ & \sum_{i \in NV} x_i \leq 250, & & \text{limite óleos não vegetais,} \\ & 3x_o \leq \sum_{i \in O} a_i x_i \leq 6x_o, & & \text{Intervalo acidez,} \\ & \sum_{i \in O} x_i = x_o, & & \text{Óleo total,} \\ & x_o, x_i \geq 0, & & \forall i \in O. \end{aligned}$$

Solução do exercício 1.11.
Sejam x_A, x_B e x_C o número de horas investidos para cada disciplina. Vamos usar variáveis auxiliares n_A, n_B e n_C para as notas finais das três disciplinas.

Como isso temos o programa linear

$$
\begin{aligned}
\textbf{maximiza} \quad & n_A + n_B + n_C, && \\
\textbf{sujeito a} \quad & x_A + x_B + x_C = 100, && \text{Total de estudo,} \\
& n_A = (6 + x_A/10)/2, && \text{Nota final disc. A,} \\
& n_B = (7 + 2x_B/10)/2, && \text{Nota final disc. B,} \\
& n_C = (10 + 3x_C/10)/2, && \text{Nota final disc. C,} \\
& n_A \geq 5, && \text{Nota mínima disc. A,} \\
& n_B \geq 5, && \text{Nota mínima disc. B,} \\
& n_C \geq 5, && \text{Nota mínima disc. C,} \\
& n_A \leq 10, && \text{Nota máxima disc. A,} \\
& n_B \leq 10, && \text{Nota máxima disc. B,} \\
& n_C \leq 10, && \text{Nota máxima disc. C,} \\
& n_A, n_B, n_C \geq 0. &&
\end{aligned}
$$

Solução do exercício 1.12.
Sejam $r \in \mathbb{R}$ e $f \in \mathbb{R}$ o número de canecos do Duff regular e do Duff Forte, respectivamente, encomendados por semana. Com isso podemos formular

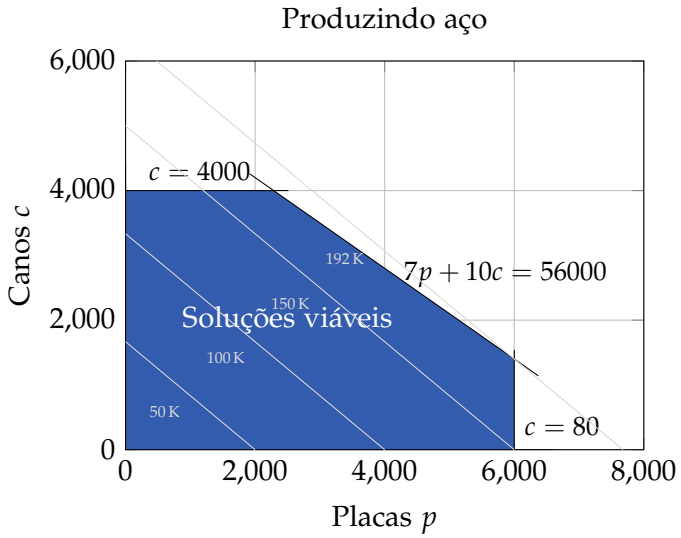
$$
\begin{aligned}
\textbf{maximiza} \quad & r + 1.5f, \\
\textbf{sujeito a} \quad & 2f \leq r, \\
& r + f \leq 3000, \\
& r, f \in \mathbb{R}_+.
\end{aligned}
$$

Solução do exercício 1.13.
Sejam $f \in \mathbb{R}$ e $h \in \mathbb{R}$ o número de pacotes de Frisky Pup e Husky Hound produzidos, respectivamente. Com isso podemos formular

$$
\begin{aligned}
\textbf{maximiza} \quad & 1.6f + 1.4h, \\
\textbf{sujeito a} \quad & f + 2h \leq 240000, \\
& 1.5f + h \leq 180000, \\
& f \leq 110000, \\
& f, h \in \mathbb{R}_+.
\end{aligned}
$$

Solução do exercício 1.14.
Sejam p e c o número de toneladas de placas e canos produzidos.

$$\begin{aligned}
\textbf{maximiza} \quad & 25p + 30c, \\
\textbf{sujeito a} \quad & p/200 + c/140 \leq 40, \iff 7p + 10c \leq 56000 \\
& p \leq 6000, \\
& c \leq 4000, \\
& c, p \geq 0.
\end{aligned}$$



A solução ótima é $p = 6000$, $c = 1400$ com valor 192000.

Solução do exercício 1.15.
Usamos índices 1, 2 e 3 para os vôos Pelotas–Porto Alegre, Porto Alegre–Torres e Pelotas–Torres e variáveis a_1, a_2, a_3 para a categoria A, b_1, b_2, b_3 para categoria B e $c-1, c_2, c_3$ para categoria C. A função objetivo é maximizar o lucro

$$z = 600a_1 + 320a_2 + 720a_3 + 440b_1 + 260b_2 + 560b_3 + 200c_1 + 160c_2 + 280c_3.$$

Temos que respeitar os limites de capacidade

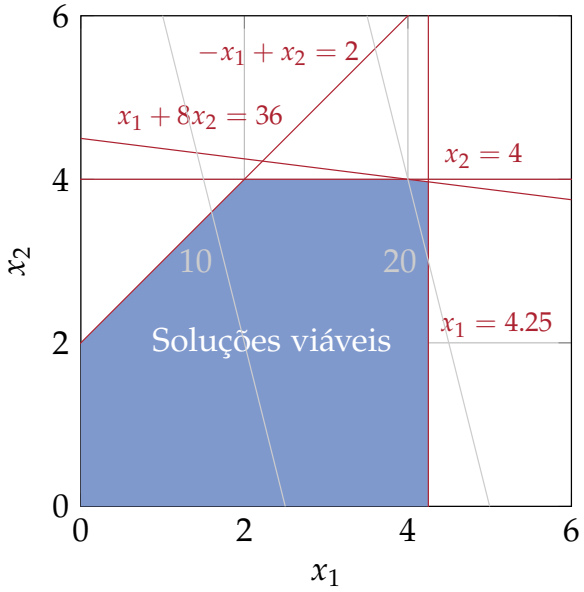
$$\begin{aligned}
a_1 + b_1 + c_1 + a_3 + b_3 + c_3 \leq 30, \\
a_2 + b_2 + c_2 + a_3 + b_3 + c_3 \leq 30,
\end{aligned}$$

e os limites da predição

$$\begin{array}{lll} a_1 \leq 4, & a_2 \leq 8, & a_3 \leq 3, \\ b_1 \leq 8, & b_2 \leq 13, & b_3 \leq 10, \\ c_1 \leq 22, & c_2 \leq 20, & c_3 \leq 18 \end{array}$$

Obviamente, todas variáveis também devem ser positivos.

Solução do exercício 1.16.
A solução gráfica é



(a) A solução ótima é $x_1 = 4.25$, $x_2 \approx 4$ (valor exato $x_2 = 3.96875$).

(b) O valor da solução ótima é ≈ 21 (valor exato 20.96875).

Solução do exercício 1.17.

$$\begin{array}{rl} \textbf{maximiza} & z = 5x_1 + 5x_2 + 5x_3 \\ \textbf{sujeito a} & -6x_1 - 2x_2 - 9x_3 \leq 0, \\ & -9x_1 - 3x_2 + 3x_3 \leq 3, \\ & 9x_1 + 3x_2 - 3x_3 \leq -3, \\ & x_1, x_2, x_3 \geq 0. \end{array}$$

$$
\begin{aligned}
\textbf{maximiza} \quad & z = -6x_1 - 2x_2 - 6x_3 + 4x_4 + 4x_5 \\
\textbf{sujeito a} \quad & -3x_1 - 8x_2 - 6x_3 - 7x_4 - 5x_5 \leq 3, \\
& 3x_1 + 8x_2 + 6x_3 + 7x_4 + 5x_5 \leq -3, \\
& 5x_1 - 7x_2 + 7x_3 + 7x_4 - 6x_5 \leq 6, \\
& x_1 - 9x_2 + 5x_3 + 7x_4 - 10x_5 \leq -6, \\
& -x_1 + 9x_2 - 5x_3 - 7x_4 + 10x_5 \leq 6, \\
& x_1, x_2, x_3, x_4, x_5 \geq 0.
\end{aligned}
$$

$$
\begin{aligned}
\textbf{maximiza} \quad & z = 7x_1 + 4x_2 + 8x_3 + 7x_4 - 9x_5 \\
\textbf{sujeito a} \quad & -4x_1 - 1x_2 - 7x_3 - 8x_4 + 6x_5 \leq -2, \\
& 4x_1 + x_2 + 7x_3 + 8x_4 - 6x_5 \leq 2, \\
& -x_1 - 4x_2 - 2x_3 - 2x_4 + 7x_5 \leq 7, \\
& -8x_1 + 2x_2 + 8x_3 - 6x_4 - 7x_5 \leq -7, \\
& 8x_1 - 2x_2 - 8x_3 + 6x_4 + 7x_5 \leq 7, \\
& x_1, x_2, x_3, x_4, x_5 \geq 0.
\end{aligned}
$$

$$
\begin{aligned}
\textbf{maximiza} \quad & z = 6x_1 - 5x_2 - 8x_3 - 7x_4 + 8x_5 \\
\textbf{sujeito a} \quad & -5x_1 - 2x_2 + x_3 - 9x_4 - 7x_5 \leq 9, \\
& 5x_1 + 2x_2 - x_3 + 9x_4 + 7x_5 \leq -9, \\
& 7x_1 + 7x_2 + 5x_3 - 3x_4 + x_5 \leq -8, \\
& -7x_1 - 7x_2 - 5x_3 + 3x_4 - x_5 \leq 8, \\
& -5x_1 - 3x_2 - 5x_3 + 9x_4 + 8x_5 \leq 0, \\
& x_1, x_2, x_3, x_4, x_5 \geq 0.
\end{aligned}
$$

Solução do exercício 2.1.
Solução com método Simplex, escolhendo como variável entrante sempre

aquela com o maior coeficiente positivo (em negrito):

$$\begin{array}{lll} z = & 25p & +30c \\ \hline w_1 = 56000 & -7p & -10c \\ w_2 = 6000 & -p & \\ w_3 = 4000 & & \mathbf{-c} \end{array}$$

$$\begin{array}{lll} z = 120000 & +25p & -30w_3 \\ \hline w_1 = 16000 & \mathbf{-7p} & +10w_3 \\ w_2 = 6000 & -p & \\ c = 4000 & & -w_3 \end{array}$$

$$\begin{array}{lll} z = 1240000/7 & -25/7p & +40/7w_3 \\ \hline p = 16000/7 & -1/7w_1 & +10/7w_3 \\ w_2 = 26000/7 & +1/7w_1 & \mathbf{-10/7w_3} \\ c = 4000 & & -w_3 \end{array}$$

$$\begin{array}{lll} z = 192000 & -3w_1 & -4w_2 \\ \hline p = 6000 & & -w_2 \\ w_3 = 2600 & +1/10w_1 & -7/10w_2 \\ c = 1400 & -1/10w_1 & +7/10w_2 \end{array}$$

Solução do exercício 2.3.
Temos

$$\binom{2(n+1)}{n+1} = \binom{2n}{n}\frac{(2n+2)(2n+1)}{(n+1)^2} = \binom{2n}{n}\frac{2(2n+1)}{n+1}$$

e logo

$$\frac{2^2 n}{n+1}\binom{2n}{n} \leq \binom{2(n+1)}{n+1} \leq 2^2\binom{2n}{n}.$$

Logo, por indução $(1/2n)2^{2n} \leq \binom{2n}{n} \leq 2^{2n}$.

Solução do exercício 2.6.

(a) Substituindo x_1 e x_2 obtemos a nova função objetivo $z = x_1 + 2x_2 = 22 - 7w_2 - 3w_1$. Como todos coeficientes são negativos, a solução básica atual permanece ótima.

(b) A nova função objetivo é $1 - w_2$ e o sistema mantem-se ótimo.

(c) A nova função objetivo é $2 - 2w_2$ e o sistema mantem-se ótimo.

(d) O dicionário dual é

$$\begin{array}{llll} z^* & = 31 & -7z_2 & -8z_1 \\ \hline y_2 & = 11 & +2z_2 & +3z_1 \\ y_1 & = 4 & +z_2 & +z_1 \end{array}$$

e a solução dual ótima é $(y_1\, y_2)^t = (4\, 11)^t$.

Solução do exercício 2.9.
Não, porque nessa situação o valor da variável entrante aumento para um valor $x_e > 0$ e por definição de variável entrante temos $c_e > 0$, i.e. o valor da função objetivo aumenta.

Solução do exercício 2.10.
Sim. Supõe que x_s, $s \in \mathcal{B}$ é a variável básica negativa. Com $x_s = \bar{b}_s - \bar{a}_{se}x_e$ e $a_{se} < 0$ temos $x_s > 0$ caso $x_e > b_s/\bar{a}_{se}$. Logo para $x_e > \max_{i\in\mathcal{B},\bar{b}_s<0} \bar{b}_i/\bar{a}_{ie}$ a solução é factível.

Solução do exercício 3.1.

$$\begin{aligned} \textbf{maximiza} \quad & 10y_1 + 6y_2 \\ \textbf{sujeito a} \quad & y_1 + 5y_2 \leq 7, \\ & -y_1 + 2y_2 \leq 1, \\ & 3y_1 - y_2 \leq 5, \\ & y_1, y_2 \geq 0. \end{aligned}$$

Solução do exercício 3.2.
Com variáveis duais y_e para cada $e \in U$ temos

$$\begin{aligned} \textbf{maximiza} \quad & \sum_{e\in U} y_e & & \\ \textbf{sujeito a} \quad & \sum_{e:e\in S} y_e \leq c(S), & & S \in \mathcal{S}, \\ & y_e \geq 0, & & e \in U. \end{aligned}$$

Solução do exercício 3.3.

(a) Temos $\mathcal{B} = \{4, 1, 2\}$ (variáveis básicas x_4, x_1 e x_2) e $\mathcal{N} = \{5, 6, 3\}$ (variáveis nulas x_5, x_6 e x_3). No que segue, vamos manter essa ordem das variáveis em todos vetores e matrizes. O vetor de custos nessa ordem é

$$c_B = (0\ 2\ -1)^t; \qquad c_N = (0\ 0\ 1)^t$$

e com

$$\Delta c = (0\ 1\ 0\ 0\ 0\ 0)^t$$

temos

$$\begin{aligned}\Delta y_N^* &= (B^{-1}N)^t \Delta c_B - \Delta c_N = (B^{-1}N)^t \Delta c_B \\ &= \begin{pmatrix} -1 & 1/2 & -1/2 \\ -2 & 1/2 & 1/2 \\ 1 & 1/2 & -3/2 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1/2 \\ 1/2 \\ 1/2 \end{pmatrix}.\end{aligned}$$

Com $y_N^* = (3/2\ 1/2\ 3/2)^t$ obtemos os limites $-1 \leq t \leq \infty$ e $1 \leq c_1 \leq \infty$.

(b) Temos $\Delta x_b = B^{-1}\Delta b$ e $\Delta b = (0\ 1\ 0)^t$. Para determinar Δx_B precisamos calcular B^{-1} pela inversão de

$$B = \begin{pmatrix} 1 & 3 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{pmatrix}$$

(observe que as colunas estão na ordem de $\mathcal{B}$) que é

$$B^{-1} = \begin{pmatrix} 1 & -1 & -2 \\ 0 & 1/2 & 1/2 \\ 0 & -1/2 & 1/2 \end{pmatrix}$$

Assim $\Delta x_B = (-1\ 1/2\ -1/2)^t$, e com $x_B^* = (10\ 15\ 5)^t$ e pela definição

$$\max_{\substack{i \in \mathcal{B} \\ \Delta x_i > 0}} -\frac{x_i^*}{\Delta x_i} \leq t \leq \min_{\substack{i \in \mathcal{B} \\ \Delta x_i < 0}} -\frac{x_i^*}{\Delta x_i}$$

obtemos os limites $-30 \leq t \leq 10$ e $-20 \leq b_2 \leq 20$.

(c) Com $\hat{b} = (70\ 20\ 10)^t$ temos $B^{-1}\hat{b} = (30\ 15\ -5)^t$. Portanto, a solução básica não é mais víavel e temos que reotimizar. O novo valor da função objetivo é

$$c_B^t(B^{-1}\hat{b}) = \begin{pmatrix}0 & 2 & -1\end{pmatrix}\begin{pmatrix}30\\15\\-5\end{pmatrix} = 35$$

e temos o dicionário

$z =$	35	$-3/2x_5$	$-1/2x_6$	$-3/2x_3$
$x_4 =$	30	$+x_5$	$+2x_6$	$-x_3$
$x_1 =$	15	$-1/2x_5$	$-1/2x_6$	$-1/2x_3$
$x_2 =$	-5	$+1/2x_5$	$-1/2x_6$	$+3/2x_3$

O dicionário é dualmente viável, e após pivô x_2–x_3 temos o novo sistema ótimo

$z =$	30	$-x_5$	$-x_6$	$-x_2$
$x_4 =$	$80/3$	$+4/3x_5$	$+5/3x_6$	$-2/3x_2$
$x_1 =$	$40/3$	$-1/3x_5$	$-2/3x_6$	$-1/3x_2$
$x_3 =$	$10/3$	$-1/3x_5$	$+1/3x_6$	$+2/3x_2$

(d) Temos $\hat{c} = (0\ 3\ -2\ 0\ 0\ 3)^t$ (em ordem $\mathcal{B},\mathcal{N}$) e com isso

$$\hat{y}_N^* = (B^{-1}N)^t\hat{c}_B - \hat{c}_N = \begin{pmatrix}-1 & 1/2 & -1/2\\ -2 & 1/2 & 1/2\\ 1 & 1/2 & -3/2\end{pmatrix}\begin{pmatrix}0\\3\\-2\end{pmatrix} - \begin{pmatrix}0\\0\\3\end{pmatrix} = \begin{pmatrix}5/2\\1/2\\3/2\end{pmatrix}$$

Portanto, a solução ainda é ótima. O novo valor da função objetivo é

$$\hat{c}_B^t(B^{-1}b) = \begin{pmatrix}0 & 3 & -2\end{pmatrix}\begin{pmatrix}10\\15\\5\end{pmatrix} = 35.$$

Solução do exercício 6.2.

Conjunto independente máximo Com variáveis indicadores x_v, $v \in V$ temos o programa inteiro

$$\begin{aligned} \textbf{maximiza} \quad & \sum_{v \in V} x_v, \\ \textbf{sujeito a} \quad & x_u + x_v \leq 1, && \forall \{u, v\} \in A, \\ & x_v \in \mathbb{B}, && \forall v \in V. \end{aligned} \tag{C.1}$$

A equação C.1 garante que cada aresta possui no máximo um nó incidente.

Emparelhamento perfeito com peso máximo Sejam x_a, $a \in A$ variáveis indicadores para a seleção de cada aresta. Com isso, obtemos o programa inteiro

$$\begin{aligned} \textbf{maximiza} \quad & \sum_{a_\in A} p(a) x_a, \\ \textbf{sujeito a} \quad & \sum_{u \in N(v)} x_{\{u,v\}} = 1, && \forall v \in V, \\ & x_a \in \mathbb{B}, && \forall v \in V. \end{aligned} \tag{C.2}$$

A equação C.2 garante que cada nó possui exatamente um vizinho.

Problema de transporte Sejam x_{ij} variáveis inteiras, que correspondem com o número de produtos transportados do depósito i para cliente j. Então

$$\begin{aligned} \textbf{minimiza} \quad & \sum_{\substack{i \in [n] \\ j \in [m]}} c_{ij} x_{ij} \\ \textbf{sujeito a} \quad & \sum_{j \in [m]} x_{ij} = p_i, \quad \forall i \in [n], \quad \text{cada depósito manda todo estoque} \\ & \sum_{i \in [n]} x_{ij} = d_j, \quad \forall j \in [m], \quad \text{cada cliente recebe a sua demanda} \\ & x_{ij} \in \mathbb{Z}^+. \end{aligned}$$

Conjunto dominante Sejam x_v, $v \in V$ variáveis indicadores para seleção de vértices. Temos o programa inteiro

$$\begin{aligned}
\textbf{minimiza} \quad & \sum_{v \in V} x_v \\
\textbf{sujeito a} \quad & x_v + \sum_{u \in N(v)} x_u \geq 1, \quad \forall v \in V, \quad \text{nó ou vizinho selecionado} \\
& x_v \in \mathbb{B}, \quad \forall v \in V.
\end{aligned}$$

Solução do exercício 6.4.
Seja $d_1 d_2 \dots d_n$ a entrada, e o objetivo selecionar $m \leq n$ dígitos da entrada. Seja $x_{ij} \in \mathbb{B}$ um indicador que o dígito $i \in [n]$ da entrada seria selecionado como dígito $j \in [m]$ da saida. Então

$$\begin{aligned}
\textbf{maximiza} \quad & \sum_{i \in [n], j \in [m]} x_{ij} d_i 10^{m-j}, \\
\textbf{sujeito a} \quad & \sum_{i \in [n]} x_{ij} = 1, && \forall j \in [m], && \text{(C.3)} \\
& \sum_{j \in [m]} x_{ij} \leq 1, && \forall i \in [n], && \text{(C.4)} \\
& x_{ij} = 0, && \forall i \in [n], j \in [m], j > i, && \text{(C.5)} \\
& x_{kl} \leq 1 - x_{ij}, && \forall i, k \in [n], l, j \in [m], k > i, l < j. && \text{(C.6)}
\end{aligned}$$

A função das restrições é a seguinte:

- Restrição (C.3) garante que tem exatamente um dígito em cada posição.
- Restrição (C.4) garante que cada dígito é selecionado no máximo uma vez.
- Restrição (C.5) garante que dígito i aparece somente a partir da posição j.
- Restrição (C.4) proibe inversões.

Solução do exercício 6.5.
Existem 21 sets diferentes, cada um com consumo diferente das 9 cartas. Seja $A\mathbb{R}^{9\times 21}$ uma matriz, que contém em cada das 21 coluna o número de cartas de cada set. Além disso, seja $b \in \mathbb{R}^9$ o número de cartas disponíveis. Usando

variáveis inteiros $x \in \mathbb{Z}^{21}$ que representam o número de sets formandos de cada tipo de set diferentes, temos a formulação

$$\begin{aligned} \textbf{maximiza} \quad & \sum_{i \in [21]} x_i \\ \textbf{sujeito a} \quad & Ax \leq b, \\ & x \geq 0. \end{aligned}$$

Solução do exercício 6.6.
Cobertura por arcos

$$\begin{aligned} \textbf{minimiza} \quad & \sum_{e \in E} c_e x_e \\ \textbf{sujeito a} \quad & \sum_{u \in N(v)} x_{uv} \geq 1, && \forall v \in V, \\ & x_e \in \mathbb{B}. \end{aligned}$$

Conjunto dominante de arcos

$$\begin{aligned} \textbf{maximiza} \quad & \sum_{e \in E} c_e x_e \\ \textbf{sujeito a} \quad & \sum_{\substack{e' \in E \\ e \cap e' \neq \emptyset}} x_{e'} \geq 1, && \forall e \in E, \\ & x_e \in \mathbb{B}. \end{aligned}$$

Coloração de grafos Seja $n = |V|$; uma coloração nunca precisa mais que n cores.

$$\begin{aligned} \textbf{minimiza} \quad & \sum_{j \in [n]} c_j \\ \textbf{sujeito a} \quad & \sum_{j \in [n]} x_{vj} = 1, && \forall v \in V, && \text{(C.7)} \\ & x_{ui} + x_{vi} \leq 1, && \forall \{u, v\} \in E, i \in [n], && \text{(C.8)} \\ & nc_j \geq \sum_{v \in V} x_{vj}, && \forall j \in [n], && \text{(C.9)} \\ & x_{vi}, c_j \in \mathbb{B}. \end{aligned}$$

- Restrição (C.7) garante que todo vértice recebe exatamente uma cor.

- Restrição (C.8) garante que vértices adjacentes recebem cores diferentes.
- Restrição (C.9) garante que $c_j = 1$ caso cor j for usada.

Clique mínimo ponderado

$$\begin{aligned} \textbf{minimiza} \quad & \sum_{v \in V} c_v x_v \\ \textbf{sujeito a} \quad & x_u + x_v \leq 1, && \forall \{u, v\} \notin E, \\ & x_v \in \mathbb{B}. \end{aligned} \tag{C.10}$$

Restrição C.10 garante que não existe um par de vértices selecionados que não são vizinhos.

Subgrafo cúbico x_e indica a seleção da aresta $e \in E$, e y_v indica se o vértice $v \in V$ ele possui grau 0 (caso contrário grau 3).

$$\begin{aligned} \textbf{minimiza} \quad & \sum_{e \in E} x_e, \\ \textbf{sujeito a} \quad & \sum_{e \in N(v)} x_e = 3y_v, && \forall v \in V, \\ & x_e \in \mathbb{B}, && \forall e \in E, \\ & y_v \in \mathbb{B}, && \forall v \in V. \end{aligned}$$

Solução do exercício 6.7.
Sejam $x_i \in \mathbb{B}, i \in [7]$ variáveis que definem a escolha do projeto i. Então temos

$$\begin{aligned} \textbf{maximiza} \quad & 17x_1 + 10x_2 + 15x_3 \\ & + 19x_4 + 7x_5 + 13x_6 + 9x_7, \\ \textbf{sujeito a} \quad & 43x_1 + 28x_2 + 34x_3 + 48x_4, \\ & + 17x_5 + 32x_6 + 23x_7 \leq 100, && \text{Limite do capital} \\ & x_1 + x_2 \leq 1, && \text{Projetos 1,2 mutualmente exclusivos} \\ & x_3 + x_4 \leq 1, && \text{Projetos 3,4 mutualmente exclusivos} \\ & x_3 + x_4 \leq x_1 + x_2, && \text{Projeto 3 ou 4 somente se 1 ou 2} \end{aligned}$$

Solução do exercício 6.8.
Seja $f \in \mathbb{B}$ uma variável que determina qual fábrica vai ser usada (fábrica 1, caso $f = 0$, fábrica 2, caso $f = 1$), $b_i \in \mathbb{B}$ uma variável binária que determina, se brinquedo i vai ser produzido e $u_i \in \mathbb{Z}$ as unidades produzidas de brinquedo i (sempre com $i \in [2]$).

$$
\begin{array}{lll}
\textbf{maximiza} & 10u_1 + 15u_2 & \\
 & \quad - 50000b_1 - 80000b_2 & \\
\textbf{sujeito a} & u_i \leq Mb_i, & \text{Permitir unidades somente se tem produção} \\
 & u_1/50 + u_2/40 \leq 500 + fM, & \text{Limite fábrica 1, se selecionada} \\
 & u_1/40 + u_2/25 \leq 700 + (1-f)M, & \text{Limite fábrica 2, se selecionada} \\
 & a_i \in \mathbb{B}, u_i \in \mathbb{Z}, i \in [3]. &
\end{array}
$$

A constante M deve ser suficientemente grande tal que ela efetivamente não restringe as unidades. Dessa forma, se a fábrica 1 está selecionada, a terceira restrição (da fábrica 2) não se aplica e vice versa.

```
http://www.inf.ufrgs.br/~mrpritt/e6q3.mod

set brinquedos := 1..2;
var f binary;
var b { brinquedos } binary;
var u { brinquedos } integer, >= 0;
param inicial { brinquedos };
param lucro { brinquedos };
param prodfab1 { brinquedos };
param prodfab2 { brinquedos };
param M := 35000;

maximize Lucro:
  sum { i in brinquedos } u[i]*lucro[i]
  - ( sum { i in brinquedos } inicial[i]*b[i] );
subject to PermitirProducao { i in brinquedos }:
  u[i] <= M*b[i];
subject to LimiteFab1 :
  sum { i in brinquedos }
    u[i]*prodfab1[i] <= 500 + f*M;
subject to LimiteFab2 :
```

```
  sum { i in brinquedos }
    u[i]*prodfab2[i] <= 700 + (1-f)*M;

data;
param inicial := 1 50000 2 80000;
param lucro := 1 10 2 15;
param prodfab1 := 1 0.020 2 0.025;
param prodfab2 := 1 0.025 2 0.040;
```

Solução: Produzir 28000 unidades do brinquedo 1 na fábrica 2, com lucro 230KR$.

Solução do exercício 6.9.
Sejam $a_i \in \mathbb{B}$ uma variável que determina se avião i vai ser produzido e $u_i \in \mathbb{Z}$ as unidades produzidas.

maximiza	$2u_1 + 3u_2 + 0.2u_3$	
	$-\ 3a_1 - 2a_2$	
sujeito a	$0.2u_1 + 0.4u_3 + 0.2u_3 \leq 1,$	Limite de capacidade
	$u_i \leq 5a_i,$	Permitir unidades somente se for produzido, limite 5 aviões
	$u_1 \leq 3,$	Limite avião 1
	$u_2 \leq 2,$	Limite avião 2
	$u_3 \leq 5,$	Limite avião 3
	$a_i \in \mathbb{B}, u_i \in \mathbb{Z}.$	

```
http://www.inf.ufrgs.br/~mrpritt/e6q4.mod

set avioes := 1..3;
param custo { avioes };
param lucro { avioes };
param capacidade { avioes };
param demanda { avioes };
var produzir { avioes } binary;
var unidades { avioes } integer, >= 0;

maximize Lucro:
```

```
  sum { i in avioes }
    (lucro[i]*unidades[i]-custo[i]*produzir[i]);
subject to LimiteCapacidade:
  sum { i in avioes } unidades[i]*capacidade[i] <= 1;
subject to PermitirProducao { i in avioes }:
  unidades[i] <= 5*produzir[i];
subject to LimiteDemanda { i in avioes }:
  unidades[i] <= demanda[i];

data;
param : custo lucro capacidade demanda :=
1 3 2   0.2 3
2 2 3   0.4 2
3 0 0.8 0.2 5;
```

Solução: Produzir dois aviões para cliente 2, e um para cliente 3, com lucro 4.8 MR$.

Solução do exercício 6.10.
Seja $x_{ijk} \in \mathbb{B}$ um indicador do teste com a combinação (i,j,k) para $1 \leq i,j,k \leq 8$. Cada combinação (i,j,k) testada cobre 22 combinações: além de (i,j,k) mais 7 para cada combinação que difere somente na primeira, segunda ou terceira posição. Portanto, uma formulação é

$$\begin{aligned}
\textbf{minimiza} \quad & \sum_{(i,j,k)\in[8]^3} x_{i,j,k} \\
\textbf{sujeito a} \quad & x_{i,j,k} + \sum_{i'\neq i} x_{i'jk} + \sum_{j'\neq j} x_{ij'k} + \sum_{k'\neq k} x_{ijk'} \geq 1, && \forall i,j,k \in [8], \\
& x_{i,j,k} \in \mathbb{B}, && \forall i,j,k \in [8].
\end{aligned}$$

A solução ótima desse sistema é 32, i.e. 32 testes são suficientes para abrir a fechadura. Uma solução é testar as combinações

$$\begin{gathered}
(1,2,4),(1,3,8),(1,5,5),(1,8,7),(2,1,1),(2,4,3),(2,6,6),(2,7,2), \\
(3,1,3),(3,4,2),(3,6,1),(3,7,6),(4,1,2),(4,4,6),(4,6,3),(4,7,1), \\
(5,1,6),(5,4,1),(5,6,2),(5,7,3),(6,2,7),(6,3,5),(6,5,4),(6,8,8), \\
(7,2,5),(7,3,7),(7,5,8),(7,8,4),(8,2,8),(8,3,4),(8,5,7),(8,8,5)
\end{gathered}$$

Solução do exercício 6.11.
Sejam $x_i \in \mathbb{B}$, $i \in [k]$ as variáveis de entrada, e $c_i \in \mathbb{B}$, $i \in [n]$ variáveis que indicam se a cláusula c_i está satisfeita. Para aplicar a regra (6.2) diretamente, vamos usar uma variável auxiliar d_i. $i \in [n]$, que representa a disjunção dos primeiros dois literais da cláusula i.

$$
\begin{aligned}
\textbf{maximiza} \quad & \sum_{i \in [n]} c_i \\
\textbf{sujeito a} \quad & l_{ij} = \begin{cases} x_k & \text{literal } j \text{ na cláusula } i \text{ é } x_k, \\ 1 - x_k & \text{literal } j \text{ na cláusula } i \text{ é } \neg x_k, \end{cases} \\
& d_i \geq (l_{i1} + l_{i2})/2, \\
& d_i \leq l_{i1} + l_{i2}, \\
& c_i \geq (d_i + l_{i3})/2, \\
& c_i \leq d_i + l_{i3}, \\
& c_i, d_i, x_i \in \mathbb{B}.
\end{aligned}
$$

Como é um problema de maximização, pode ser simplificado para

$$
\begin{aligned}
\textbf{maximiza} \quad & \sum_{i \in [n]} c_i \\
\textbf{sujeito a} \quad & l_{ij} = \begin{cases} x_k & \text{literal } j \text{ na cláusula } i \text{ é } x_k, \\ 1 - x_k & \text{literal } j \text{ na cláusula } i \text{ é } \neg x_k, \end{cases} \\
& c_i \leq l_{i1} + l_{i2} + l_{i3}, \\
& c_i, x_i \in \mathbb{B}.
\end{aligned}
$$

A segunda formulação possui uma generalização simples para o caso $k > 3$.

Solução do exercício 6.13.
Não. Uma explicação: `http://nbviewer.jupyter.org/url/www.inf.ufrgs.br/~mrpritt/oc/greedy-independent-set.ipynb`.

Solução do exercício 6.15.
Não. Primeiramente, a restrição

$$\prod_{p \in P} x_p = 10! \tag{C.11}$$

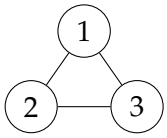
não é linear. Mas mesmo ignorando isso as restrições não definem uma bijeção entre números e posições. O conjunto completo de soluções é

$$
\begin{aligned}
&1,2,3,4,5,6,7,8,9,10\\
&1,2,3,4,6,6,6,7,10,10\\
&1,2,4,4,4,5,7,9,9,10\\
&1,3,3,3,4,6,7,8,10,10\\
&1,3,3,4,4,4,7,9,10,10\\
&2,2,2,3,4,6,7,9,10,10
\end{aligned}
$$

Solução do exercício 7.1.
Se $(A\ B)$ é TU, então trivialmente A é TU. Agora caso A é TU, considere uma submatriz quadrada $(A'\ B')$ de $(A\ B)$. Como B somente possui um coeficiente não-nulo por coluna temos $\det(A'\ B') = \pm\det(A)$. Logo $(A'\ B')$ é TU.

Solução do exercício 7.2.

Conjunto independente máximo A matriz de coeficientes contém dois coeficientes igual 1 em cada linha, que correspondem com uma aresta, mas geralmente não é totalmente unimodular. Por exemplo, o grafo completo com três vértices K_3



gera a matriz de coeficientes

$$
\begin{pmatrix}
1 & 1 & 0\\
1 & 0 & 1\\
0 & 1 & 1
\end{pmatrix}
$$

cuja determinante é -2. A solução ótima da relaxação inteira $0 \leq x_i \leq 1$ é $x_1 = x_2 = x_3 = 1/2$ com valor $3/2$, a Fig. C.1 mostra o polítopo correspondente. (Observação: A transposta dessa matriz satisfaz os critérios (i) e (ii)

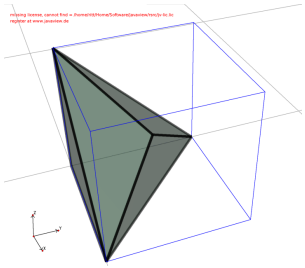


Figura C.1.: Polítopo $\{x \in \mathbb{R}^3 \mid x_1 + x_2 \leq 1, x_1 + x_3 \leq 1, x_2 + x_3 \leq 1, 0 \leq x_i \leq 1\}$. (O visualizador usa os eixos $x = x_1$, $y = x_2$, $z = x_3$.)

da nossa proposição, e caso o grafo é bi-partido, também o critério (iii). Portanto *Conjunto independente máximo* pode ser resolvido em tempo polinomial em grafos bi-partidos).

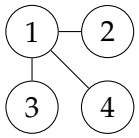
Emparelhamento perfeito com peso máximo A matriz de coeficientes satisfaz critério (i). Ela tem uma linha para cada vértice e uma coluna para cada aresta do grafo. Como cada aresta é incidente a exatamente dois vértices, ela também satisfaz (ii). Finalmente, a bi-partição $V_1 \dot\cup V_2$ do grafo gera uma bi-partição das linhas que satisfaz (iii). Portanto, a matriz é TU, e o *Emparelhamento perfeito com peso máximo* pode ser resolvido em tempo polinomial usando a relaxação linear.

Problema de transporte A matriz de coeficientes satisfaz critério (i). Podemos representar o problema como grafo bi-partido completo $K_{n,m}$ entre os depósitos e os clientes. Desta forma, com o mesmo argumento que no último problema, podemos ver, que os critérios (ii) e (iii) são satisfeitos.

Conjunto dominante A matriz de coeficientes satisfaz critério (i), mas não critério (ii): cada linha e coluna correspondente com vértice v contém $|N(v)| + 1$ coeficientes não-nulos. Mas, não é obviou se a matriz mesmo assim não é TU (lembra que o critério é suficiente, mas não necessário). O K_3 acima, por exemplo, gera a matriz

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

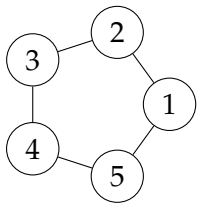
que *é* TU. Um contra-exemplo seria o grafo bi-partido $K_{1,3}$



que gera a matriz de coeficientes

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

com determinante -2. Isso não prova ainda que a relaxação linear não produz resultados inteiros ótimos. De fato, nesse exemplo a solução ótima da relaxação inteira é a solução ótima inteira $D = \{1\}$.
Um verdadeiro contra-exemplo é um ciclo com cinco vértices C_5



com matriz

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

(cuja determinante é 3). A relaxação linear desse sistema tem a solução ótima $x_1 = x_2 = x_3 = x_4 = x_5 = 1/3$ com valor $5/3$ que não é inteira.

Solução do exercício 7.4.
A formulação possui 14 restrições, correspondendo com as 14 arestas. Como o grafo é 4-regular, cada vértice ocorre 4 vezes no lado esquerdo de uma restrição, e somando todas restrições obtemos

$$\begin{aligned} & 4 \sum_{i \in [7]} x_i \leq 14 \\ \Rightarrow & \sum_{i \in [7]} x_i \leq 14/4 \\ \Rightarrow & \sum_{i \in [7]} x_i \leq \lfloor 14/4 \rfloor = 3, \end{aligned}$$

que não é suficiente. Para obter uma desigualdade mais forte, vamos somar sobre todos triângulos. Somando primeiro as restrições das arestas de cada triângulo (u, v, w) obtemos

$$\begin{aligned} & 2x_u + 2x_v + 2x_w \leq 3 \\ \Rightarrow & x_u + x_v + x_w \leq \lfloor 3/2 \rfloor = 1. \end{aligned}$$

Somando agora as restrições obtidas desta forma de todos 14 triângulos do grafo (cada vértice é parte de 6 triângulos) obtemos a desigualdade desejada

$$\begin{aligned} 6 \sum_{i \in [7]} x_i &\leq 14 \\ \Rightarrow \sum_{i \in [7]} x_i &\leq \lfloor 14/6 \rfloor = 2. \end{aligned}$$

(Outra abordagem: Supõe, sem perda de generalidade, que $x_1 = 1$ na solução ótima. Pelas restrições $x_1 + x_i \leq 2$ temos $x_i = 0$ para $i \in \{3, 4, 5, 6\}$. Pela restrição $x_2 + x_7 \leq 1$, portanto $\sum_{1 \leq i \leq 7} x_i \leq 2$.)

Solução do exercício 7.5.
Seja $\bar{S} = [n] \setminus S$ e $m = \max_{i \in S} a_i$ e $\bar{m} = \max_{i \in \bar{S}} a_i$. A idéia é somar desigualdades $x_i \leq 1$ para $i \in S$ até o corte de Gomory obtido pela divisão pelo coeficiente máximo em S rende a desigualdade desejada. Seja $\delta = \max\{\bar{m} + 1, m\}$. Somando $(\delta - a_i)x_i \leq \delta - a_i$ obtemos

$$\sum_{i \in S} \delta x_i + \sum_{i \in \bar{S}} a_i x_i \leq b + \sum_{i \in S} (\delta - a_i) x_i < \delta |S| \leq \delta |S| - 1.$$

Aplicando o corte de Gomory com multiplicador $1/\delta$ obtemos

$$\sum_{i \in S} x_i \leq \lfloor |S| - 1/\delta \rfloor = |S| - 1$$

porque $a_i \leq \bar{m} < \max\{\bar{m} + 1, m\} = \delta$ e logo $\lfloor a_i/\delta \rfloor = 0$ para $i \in \bar{S}$.

Solução do exercício 7.6.
$x_1 + x_6 + x_7 \leq 2$ porque uma rota não contém subrotas. Portanto $x_1 + x_2 + x_5 + x_6 + x_7 + x_9 \leq 5$. Supõe $x_1 + x_2 + x_5 + x_6 + x_7 + x_9 = 5$. Temos três casos: $x_1 = 0$, $x_6 = 0$ ou $x_7 = 0$. Em todos os casos, as restantes variáveis possuem valor 1, e no grafo resultante sempre existe um vértice de grau 3 (o vértice no centro, da esquerda, de acima, respectivamente), que não é possível numa solução válida.

Solução do exercício 7.8.
O sistema inicial

$z =$		x_1	$+3x_2$
$w_1 =$	-2	$+x_1$	
$w_2 =$	3		$-x_2$
$w_3 =$	-4	$+x_1$	$+x_2$
$w_4 =$	12	$-3x_1$	$-x_2$

não é primalmente nem dualmente viável. Aplicando a fase I (pivôs x_0–w_3, x_0–x_1) e depois fase II (pivôs x_2–w_1, w_3–w_2, w_1–w_4) gera o dicionário final

$$\begin{array}{rrll} z = & 12 & -8/3w_2 & -1/3w_4 \\ \hline x_2 = & 3 & -w_2 & \\ w_3 = & 2 & -2/3w_2 & -1/3w_4 \\ x_1 = & 3 & +1/3w_2 & -1/3w_4 \\ w_1 = & 1 & +1/3w_2 & -1/3w_4 \end{array}$$

cuja solução $x_1 = 3$, $x_2 = 3$ já é inteira.
No segundo sistema começamos com o dicionário

$$\begin{array}{rrll} z = & & x_1 & -2x_2 \\ \hline w_1 = & 60 & +11x_1 & -15x_2 \\ w_2 = & 24 & -4x_1 & -3x_2 \\ w_3 = & 59 & -10x_1 & +5x_2 \end{array}$$

e um pivô x_1–w_3 gera a solução ótima fracionária

$$\begin{array}{rrll} z = & 4.9 & -0.1w_3 & -1.5x_2 \\ \hline w_1 = & 113.9 & -1.1w_3 & -9.5x_2 \\ w_2 = & 4.4 & +0.4w_3 & -5x_2 \\ x_1 = & 4.9 & -0.1w_3 & +0.5x_2 \end{array}$$

e a linha terceira linha (x_1) gera o corte

$$w_4 = \quad -0.9 \quad +0.1w_3 \quad +0.5x_2$$

Com o pivô w_4–w_3 obtemos a solução ótima inteira

$$\begin{array}{rrll} z = & 4 & -w_4 & -x_2 \\ \hline w_1 = & 104 & -11w_4 & -4x_2 \\ w_2 = & 8 & +4w_4 & -7x_2 \\ x_1 = & 4 & -w_4 & +1x_2 \\ w_3 = & 9 & +10w_4 & -5x_2 \end{array}$$

Bibliografia

Anstreicher, K. M. (1999). "Linear programming in $O((n^3 \log n)L)$ operations". Em: *SIAM J. Opt.* 9.4, pp. 803–812 (ver p. 47).

Applegate, D. L., R. E. Bixby, V. Chvátal e W. J. Cook (2007). *The Traveling Salesman Problem: A Computational Study*. Princeton University Press (ver pp. 95–97).

Ausiello, G., P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela e M. Protasi (1999). *Complexity and approximation – Combinatorial Optimization Problems and their Approximability Properties*. INF 510.5 C737. Springer-Verlag. URL: http://www.nada.kth.se/~viggo/approxbook.

Cerny, V. (1985). "Thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm". Em: *J. Opt. Theor. Appl.* 45, pp. 41–51 (ver p. 170).

Clausen, J. (1999). *Branch and Bound Algorithms – Principles and examples* (ver pp. 142, 147).

Cook, W. (dez. de 2011). *Concorde TSP solver* (ver p. 147).

– (2012). "Markovitz and Manne + Eastman + Land and Doig = Branch and bound". Em: *Document Mathematica* Special volume 21st ISMP, pp. 227–238 (ver p. 147).

Dakin, R. J. (1965). "A tree-search algorithm for mixed integer programming problems". Em: *The Computer Journal* 8.3, pp. 250–255. DOI: 10.1093/comjnl/8.3.250 (ver p. 147).

Dasgupta, S., C. Papadimitriou e U. Vazirani (2009). *Algoritmos*. McGraw-Hill (ver p. 23).

Fearnley, J. e R. Savani (2014). "The Complexity of the Simplex Method". Em: *Arxiv* (ver p. 48).

Garey, M. R. e D. S. Johnson (1979). *Computers and intractability: A guide to the theory of NP-completeness*. Freeman (ver p. 147).

Ghouila-Houri, A. (1962). "Caractérisation des matrices totalement unimodulaires". Em: *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences* 254, pp. 1192–1194 (ver p. 127).

Hoffman, A. J. e J. B. Kruskal (1956). "Linear Inequalities and Related Systems". Em: ed. por H. W. Kuhn e A. J. Tucker. Princeton University Press. Cap. Integral boundary points of convex polyhedra, pp. 223–246 (ver p. 129).

Karp., R. M. (1972). "Reducibility Among Combinatorial Problems". Em: *Complexity of Computer Computations*. Ed. por R. E. Miller e J. W. Thatcher. New York: Plenum, pp. 85–103 (ver p. 112).

Kirkpatrick, S., C. D. Gelatt e M. P. Vecchi (1983). "Optimization by simulated annealing". Em: *Science* 220, pp. 671–680 (ver p. 170).

Land, A. H. e A. G. Doig (1960). "An automatic method of solving discrete programming problems". Em: *Econometrica* 28.3, pp. 497–520. DOI: 10.2307/1910129 (ver p. 141).

Maculan, N. e M. H. C. Fampa (2006). *Otimização linear*. INF 65.012.122 M175o. Editora UnB (ver p. 48).

Metropolis, N., A. Rosenbluth, M. Rosenbluth, A. Teller e E. Teller (1953). "Equation of state calculations by fast computing machines". Em: *Journal of Chemical Physics* 21, pp. 1087–1092 (ver p. 168).

Nemhauser, G. L. e L. A. Wolsey (1999). *Integer and Combinatorial Optimization*. Wiley. DOI: 10.1002/9781118627372 (ver pp. 132, 136).

Ruiz, R. e T. Stützle (2007). "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem". Em: *Eur. J. Oper. Res.* 177.3, pp. 2033–2049. DOI: 10.1016/j.ejor.2005.12.009 (ver p. 181).

Spielman, D. A. e S. H. Teng (mai. de 2004). "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time". Em: *J. ACM* 51.3, pp. 385–463. ISSN: 0004-5411. DOI: 10.1145/990308.990310. URL: http://dx.doi.org/10.1145/990308.990310 (ver p. 47).

Truemper, K. (1990). "A decomposition theory for matroids. V. Testing of matrix total unimodularity". Em: *J. Comb. Theory, Ser. B* 49, pp. 241–281 (ver p. 147).

Vanderbei, R. J. (2014). *Linear programming: Foundations and Extensions*. 4th. INF 65.012.122 V228l. Kluwer. DOI: 10.1007/978-1-4614-7630-6. URL: http://www.princeton.edu/~rvdb/LPbook (ver pp. 23, 24, 47).

Williams, H. P. (1986). "Fourier's method of linear programming and its dual". Em: *The American Mathematical Monthly* 93.9, pp. 681–695 (ver p. 18).

Wolsey, L. A. (1998). *Integer programming*. Wiley (ver p. 136).

Índice