Parte II

Programação inteira

6 Introdução

6.1 Definições

Problema da dieta

- Problema da dieta

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

- com limites quantidade de comida x.
- 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?

- Única refeição? Como resolver?
- Restringe a variáveis x ao $\mathbb{Z}$.
- Será que metodo 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)

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

- Programação inteira pura (PI)

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

Programação inteira

- Programção (inteira) mista (PIM)

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

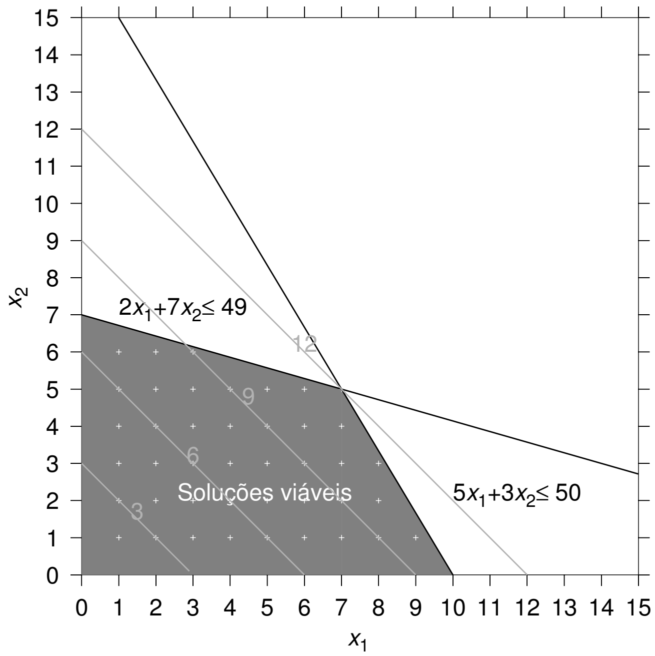
- Programação linear e inteira pura são casos particulares da programação mista.
- Outro caso particular: 0-1-PIM e 0-1-PI.

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

Exemplo

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

Exemplo

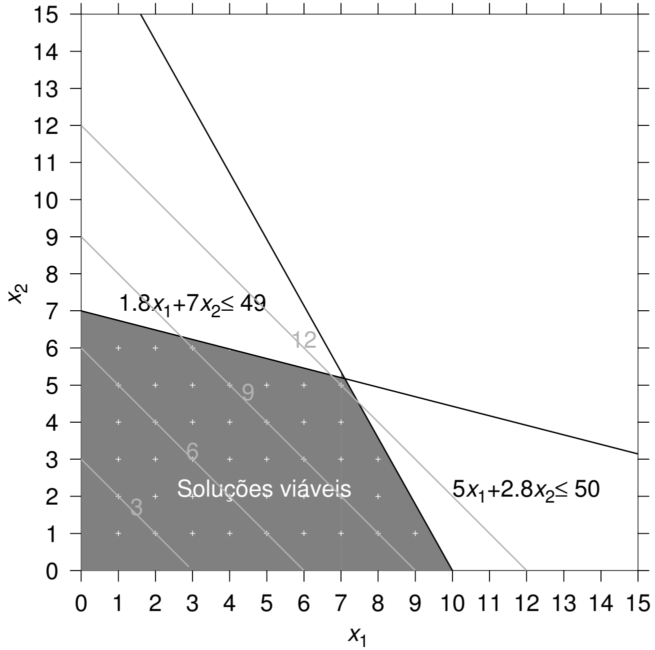


- Sorte: A solução ótima é inteira! $x_1 = 7$, $x_2 = 5$, $V = 12$.
- Observação: Se a solução ótima é inteira, um problema de PI(M) pode ser resolvido com o método Simplex.

Exemplo

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

Exemplo

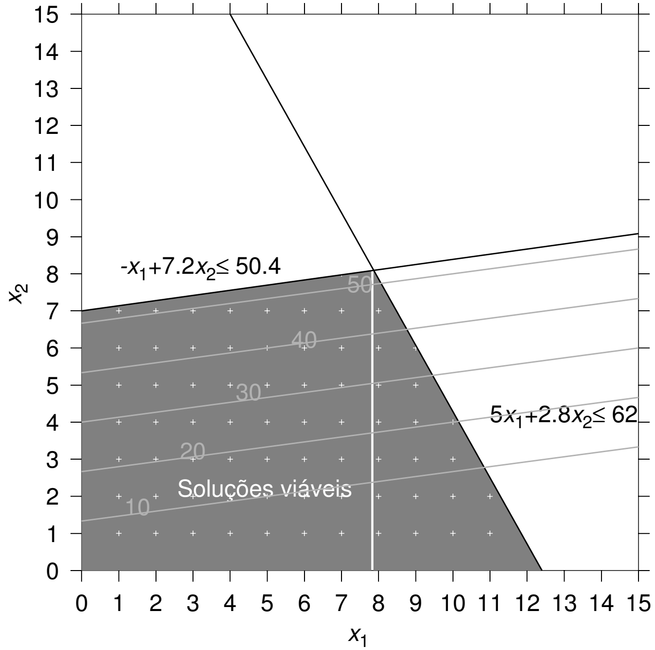


- Solução ótima agora: $x_1 \approx 7.10$, $x_2 \approx 5.17$, $V = 12.28$.
- Será que $\lfloor x_1 \rfloor, \lfloor x_2 \rfloor$ é a solução ótima do PI?

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 arbitrairamente distante!

Métodos

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

Exemplo: 0-1-**Knapsack**

PROBLEMA DA MOCHILA (KNAPSACK)

Instância Um conjunto de n items $I = \{i_1, \ldots, i_n\}$ com valores v_i e pesos w_i. Um limite de peso K do mochila.

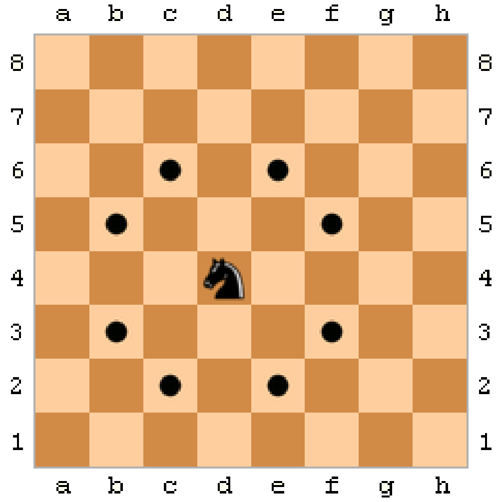
Solução Um conjunto $S \subseteq I$ de elements que cabem na mochila, i.e. $\sum_{i \in S} w_i \leq K$.

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

- Observação: Existe um solução com programação dinâmica que possui complexidade de tempo $O(Kn)$ (pseudo-polinomial) e de espaço $O(K)$.

Exemplo: Maximizar cavalos

- Qual o número máximo de cavalos que cabe num tabuleiro de xadrez, tal que nenhum ameaça um outro?



Exemplo 6.1
Formulação do problema da mochila, com variáveis indicadores x_i, $1 \leq i, j \leq$

8:

$$\begin{aligned} \textbf{maximiza} \quad & \sum_i v_i x_i \\ \textbf{sujeito a} \quad & \sum_i w_i x_i \leq L \\ & x_i \in \mathbb{B} \end{aligned}$$

Formulação do problema dos cavalos com variáveis indicadores x_{ij}:

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

Soluções do problema dos cavaleiros (A030978)

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
k	1	4	5	8	13	18	25	32	41	50	61	72	85	98	113	128

◊

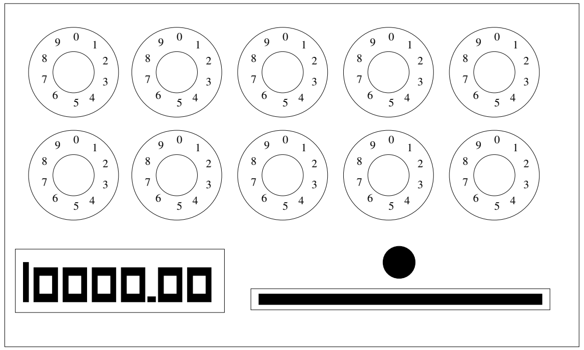
6.2 Motivação e exemplos

Motivação

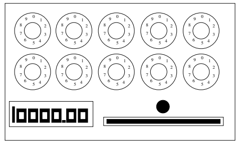
- Otimização combinatória é o ramo da ciência da computação que estuda problemas de otimização em conjuntos (wikipedia).
- "The discipline of applying advanced analytical methods to help make better decisions" (INFORMS)
- Tais problemas são extremamente frequentes e importantes.

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:$\Rightarrow 10 \times 60 \times 60 \times 24 \times 365 \cong 3 \times 10^8$

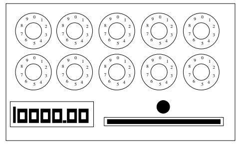
Explosão combinatória

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"

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



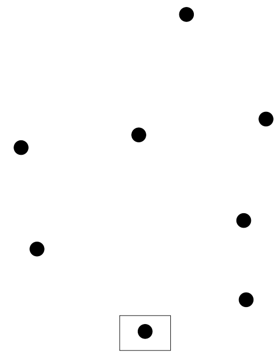
- Melhor não aceitar a máquina de dinheiro.
- Problemas combinatórios são difíceis.

6.3 Aplicações

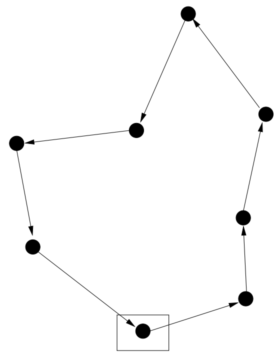
Apanhado de problemas de otimização combinatória

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

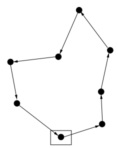
Caixeiro Viajante



Caixeiro Viajante



Caixeiro Viajante



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

Caixeiro Viajante

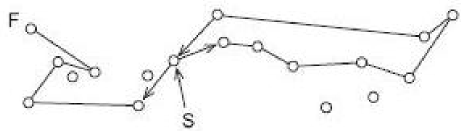


Figure 1.40 Chimpanzee tour (Bido).

Caixeiro Viajante

[1]Retirado de: "The Traveling Salesman Problem: A Computational Study" David L. Applegate, Robert E. Bixby, Vasek Chvátal & William J. Cook. Princeton University Press

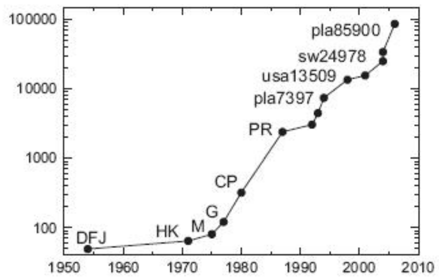
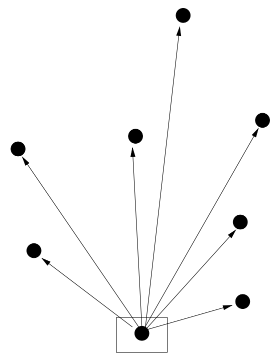


Figure 1.45 Further progress in the TSP, log scale.

Formulando matemáticamente o PCV

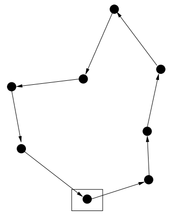
- Associar uma variável a cada possível decisão.



Formulando matemáticamente o PCV

- Associar uma variável a cada possível decisão.

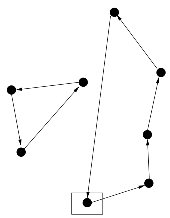
[1]Retirado de: "The Traveling Salesman Problem: A Computational Study" David L. Applegate, Robert E. Bixby, Vasek Chvátal & William J. Cook. Princeton University Press



$$\begin{array}{lll}\textbf{minimiza} & c_{ij}y_{ij} & \\ \textbf{sujeito a} & \sum_{j\in N} x_{ij} + \sum_{j\in N} x_{ji} = 2, & \forall i \in N \\ & x_{ij} \in \{0,1\}, & \forall i,j \in N.\end{array}$$

Formulando matemáticamente o PCV

- Associar uma variável a cada possível decisão.



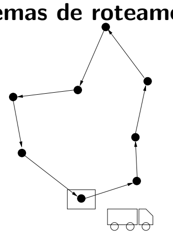
$$\begin{array}{lll}\textbf{minimiza} & c_{ij}y_{ij} & \\ \textbf{sujeito a} & \sum_{j\in N} x_{ij} + \sum_{j\in N} x_{ji} = 2, & \forall i \in N \\ & x_{ij} \in \{0,1\}, & \forall i,j \in N.\end{array}$$

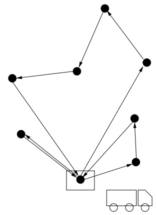
+ restrições de eliminação de subciclos!

Apanhado de problemas de otimização combinatória

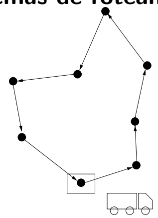
- Caixeiro viajante
- Roteamento
- Projeto de redes
- Alocação de horários
- Tabelas esportivas
- Gestão da produção
- etc.

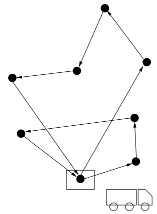
Problemas de roteamento

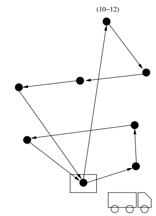


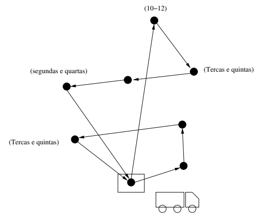


Problemas de roteamento







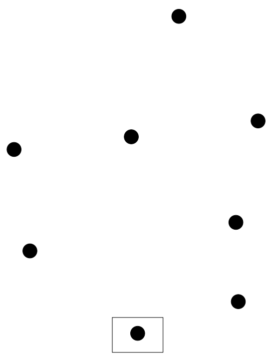


Etc.

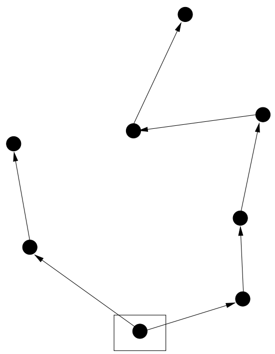
Apanhado de problemas de otimização combinatória

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

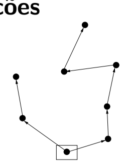
Problemas em árvores



Problemas em árvores



Problemas em árvores - aplicações



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

Apanhado de problemas de otimização combinatória

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

Alocação de tripulações

Apanhado de problemas de otimização combinatória

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

Tabelas esportivas

Proximos Adversários

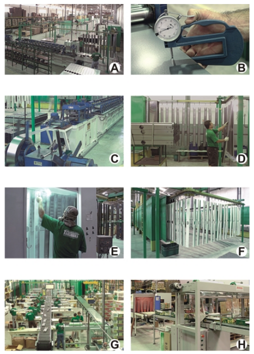
Fla	Vasco	Paysandu	Criciuma	Vitória
JUVENTUDE	Ponte	Coritiba	GALO	CORINTHIANS
Guarani	CRUZEIRO	PALMEIRAS	Santos	Juventude
GALO	São Paulo	Paraná	FURACÃO	GUARANI
Botafogo	GOIÁS	CRICIÚMA	Paysandu	Grêmio
PALMEIRAS	Juventude	Santos	PONTE	COXA
Coritiba	CORINTHIANS	GALO	Paraná	São Paulo
S. PAULO	Furacão	Guarani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

Apanhado de problemas de otimização combinatória

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

Gestão da produção



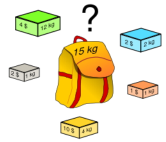
7 Formulação

7.1 Exemplos

"Regras de formulação"

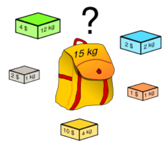
- Criar (boas) formulações é uma arte.
- Algumas diretivas básicas:
  - escolha das variáveis **de decisão**.
  - escolha do objetivo.
  - ajuste das restrições.

Formulação - Problema da mochila



- itens $N = \{1, 2, ...n\}$
- peso de cada ítem: p_i, valor de cada ítem: v_i
- Levar o maior valor possível, dada a restrição de peso.
- Variáveis de decisão ?

Formulação - Problema da mochila



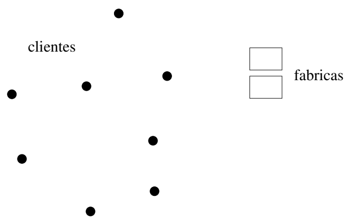
$$\text{Max } v_i x_i$$

s.t.

$$\sum_{i \in N} p_i x_i \leq P$$
$$x_i \in \{0,1\}$$

Formulação - Problema de locação de facilidades não-capacitado

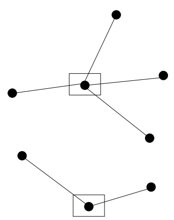
- Alocar fábricas a cidades, de modo a minimizar o custo total de instalação das fábricas e custo de transporte do produto até o cliente



- Cada ponto $i = \{1, 2, ...n\}$ apresenta um custo de instalação da fábrica f_i
- Entre cada par de cidade, (i, j), o custo de transporte é dado por c_{ij}

Formulação - Problema de locação de facilidades não-capacitado

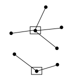
- Exemplo:



- Variáveis de decisão ?

Para formulação escolhemos variáveis de decisão $x_{ij} \in \mathbb{B}$, que indicam se o cliente i for atendido pela fábrica em j.

Formulação - Problema de locação de facilidades não-capacitado



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

Alternativas:

- Para instalar exatamente m fábricas: $\sum y_j = m$.

7.2 Técnicas

Formulação: Indicadores

- Variáveis indicadores $x \in \mathbb{B}$: 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:

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

- Ou-exlusivo:

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

- Em geral: Seleciona n de m itens $x_1, \ldots, x_m \in \mathbb{B}$

$$\sum_i x_i \begin{Bmatrix} = \\ \geq \end{Bmatrix} n$$

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

- Conjunção $x = yz = y \wedge z$

$$x \le (y+z)/2$$
$$x \ge y + z - 1$$

- Disjunção $x = y \vee z$

$$x \ge (y+z)/2$$
$$x \le y + z$$

- Negação $x = \neg y$

$$x = 1 - y$$

Formulação: Função objetivo não-linear

- Queremos minimizar custos, com uma “entrada” fixa c

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

  com $l(x)$ linear.

- Solução?

$$f(x) = cy + l(x)$$
$$x \le \bar{x}y$$
$$x \in R, y \in \mathbb{B}$$

- Disjunção de equações: Queremos que aplica-se uma das equações

$$f_1 \le f_2$$
$$g_1 \le g_2$$

- Solução, com constante M suficientemente grande

$$f_1 \le f_2 + Mx$$
$$g_1 \le g_2 + M(1-x)$$
$$x \in \mathbb{B}$$

Exemplo
Planejamento de produção (ingl. uncapacitated lot sizing)

- Objetivo: Planejar a futura produção no próximos n semanas.

- Parametros: Para cada semana i
  - 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 produzido,

- s_i a quantidade no estoque no final da semana i,

- $y_i = 1$ sem tem produção na semana i, 0 senão.

Problema:

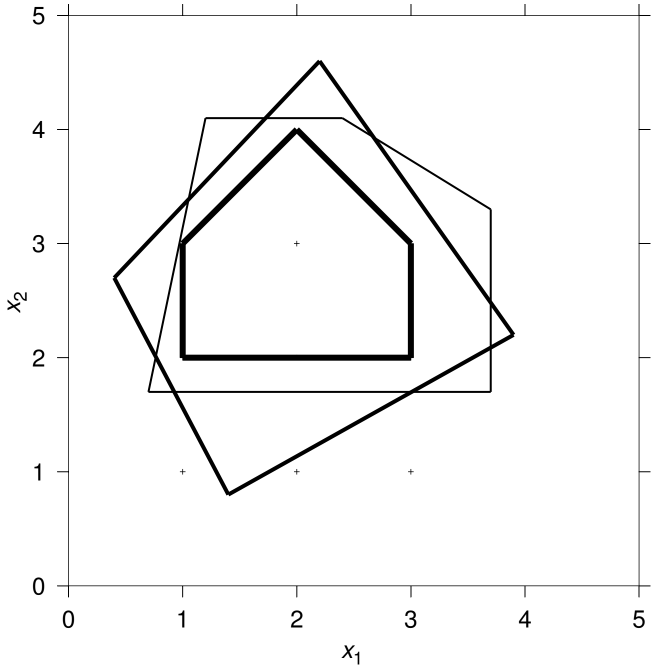
- Função objetivo tem custos fixos, mas x_i não tem limite.

- Determina ou estima um valor limite M.

Exemplo

$$\begin{array}{lll} \textbf{minimiza} & \sum_i p_i x_i + \sum_i h_i s_i + \sum_i f_i y_i & \\ \textbf{sujeito a} & s_i = s_{i-1} + x_i - d_i, & 1 \le i \le n \\ & s_0 = 0 & \\ & x_i \le M y_i, & 1 \le i \le n \\ & x \in \mathbb{R}^n, y \in \mathbb{B}^n. & \end{array}$$

Formulações diferentes



As formulações acima são limitadas. Por exemplo, podemos escrever $x \leq y$ para formular uma implicação $x \to y$ para duas variáveis booleanas. Isso não tem generalização simples para outros casos. Por exemplo qual seria uma formulação de

$$x_1 \to x_2 + x_3 \leq 4$$
$$x_1 \in \mathbb{B}, x_2, x_3 \in \mathbb{R},$$

ou seja, a equação $x_2 + x_3 \leq 4$ só precisa ser satisfeita, quando a variável booleana x_1 é verdadeira? Uma abordagem natural é

$$x_1(x_2 + x_3) \leq 4 \Leftrightarrow x_1x_2 + x_1x_3 \leq 4$$

mas, infelizmente, gera uma restrição *quadrática*, porque contém produtos de variáveis como x_1x_3. Como programas quadráticas são consideravelmente mais complicados de resolver, o nosso objetivo é formular problemas *lineares*. Logo, a formulação acima não serve.

Caso conhecemos limites superiores

$$x_2 \leq u_2$$
$$x_3 \leq u_3$$

sabemos também que $x_2 + x_3 \leq u_2 + u_3$. TBD TBD

$$x_2 + x_3 \leq 4 + (1 - x_1)M$$

com $M := u_2 + u_3$.

8 Técnicas de solução

8.1 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): Essentialmente usando uma relaxação
  - Menos restrições $\Rightarrow$ conjunto maior de solução viáveis.
  - Nova função objetivo que é maior ou igual.
- Importante: Relaxação linear: $x \in \mathbb{Z} \Rightarrow x \in \mathbb{R}$.

8.2 Problemas com solução eficiente

Relaxação inteira

- 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.

Lembrança: Determinante usando Laplace

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

com A_{ij} a submatriz sem linha i e coluna j.

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-gésima coluna de A por b.

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

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

É simples verificar que $AU_i = A_i$. Com $\det(U_i) = x_i$ e $\det(A)\det(U_i) = \det(A_i)$ temos o resultado. ■

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

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

$$\begin{vmatrix} 1 & 2 & 1 \\ 1 & 0 & 2 \\ 1 & 1 & 2 \end{vmatrix} = -1$$

$$\begin{vmatrix} 3 & 1 & 1 \\ 5 & 1 & 2 \\ 2 & 1 & 2 \end{vmatrix} = -3$$

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

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

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 quadrática não-singular de A tem determinante $\{-1, 1\}$.

Exemplo 8.1
Observe que essas condições são suficientes, mas não necessárias. É possível que $Bx = b$ possui solução inteira sem essas condições ser satisfeitas. Por exemplo

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

tem a solução inteira $(x_1 x_2) = (10)$, mesmo que $\det(A) = -2$. ◊

A relaxação é inteira

Definição 8.1
Uma matriz quadrática 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 unimodular?

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

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

Critérios

Proposição 8.1
Se A é TU então

1. A^t é TU.
2. $(A\ I)$ com matriz de identidade I é TU.
3. Uma matriz B que é uma permutação das linhas ou colunas de A é TU.
4. Multiplicando uma linha ou coluna com -1 resulta numa 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 determinate troca no máximo o sinal. ■

Critérios

Proposição 8.2
Uma matriz A é totalmente unimodular se

1. $a_{ij} \in \{+1, -1, 0\}$
2. Cada coluna contém no máximo dois coeficientes não-nulos.
3. Existe uma partição de linhas $M_1 \dot\cup M_2 = [1, m]$ tal que cada coluna *com dois coeficientes não-nulos* satisfaz

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

Observe que esse critério é suficiente, mas não necessário.

Exemplo

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

- Coeficientes $\in \{-1, 0, 1\}$: Sim.
- Cada coluna no máximo dois coeficientes não-nulos: Sim.
- Partição M_1,M_2? Sim, escolhe $M_1 = [1, 3], M_2 = \emptyset$.

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: $\det(A) = 2$.

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

TU? Sim. Mas nossa regra não se aplica!
Prova. (Proposição 8.2). Prova por contradição. Seja A uma matriz que satisfaz os critérios da proposição 8.2, e seja B o menor submatriz quadrada de A tal que $\det(B) \notin \{0, +1, -1\}$. B não contém uma coluna com um único coeficiente não-nula: 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. ■

Consequências

Teorema 8.1 (Hoffman,Kruskal)
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

$$\{x \in \mathbb{R}^n | Ax \le b\}$$
$$\{x \in \mathbb{R}^n | Ax \ge b\}$$
$$\{x \in \mathbb{R}^n | Ax \le b, x \ge 0\}$$
$$\{x \in \mathbb{R}^n | Ax = b, x \ge 0\}$$

tem pontos extremos inteiros.

Prova. Considerações acima. ■

Exemplo 8.2 (Caminhos mais curtos)

Exemplo: Caminhos mais curtos

- Dado um grafo não-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

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

A matriz do sistema acima de forma explicita:

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

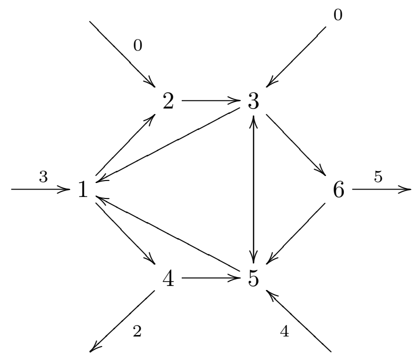
Como cada arco é adjacente ao no máximo dois vértices, e cada coluna contém um coeficiente 1 e -1, a Proposição 8.2 é satisfeito com a partição trivial. $\Diamond$

Exemplo 8.3 (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?



Exemplo: Fluxo em redes

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

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

Exemplo: Fluxo

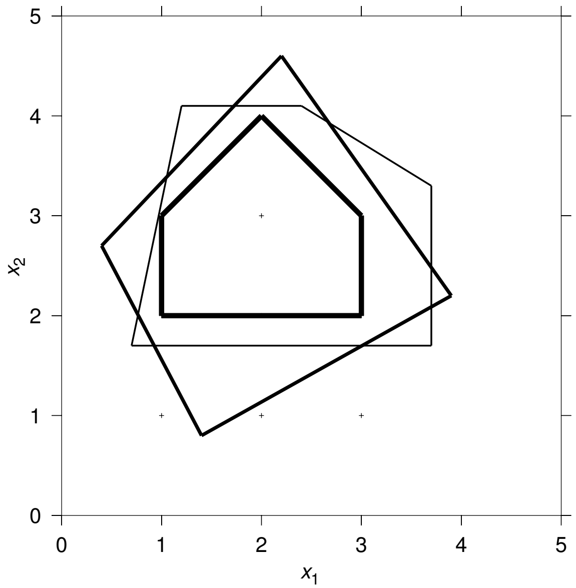
- A matriz que define um problema de fluxo é totalmente unimodular.
- Consequências
  - Cada ponto extremo da região víavel é inteira.
  - A relaxação PL resolve o problema.
- Existem vários subproblemas de fluxo mínimo que podem ser resolvidos também, p.ex. fluxo máximo entre dois vértices.

$\Diamond$

8.3 Desigualdades válidas

Desigualdades válidas

- Problema inteiro
$$\max\{c^t x | Ax \le b, x \in Z_+^n\}$$
- Relaxação linear
$$\max\{c^t x | Ax \le b, x \in R_+^n\}$$



Desigualdades válidas

Definição 8.2
Uma desigualdade $\pi x \le \pi_0$ é *válida* para um conjunto P, se $\forall x \in P : \pi x \le \pi_0$.

- Como achar desigualdades (restrições) válidas para o conjunto da soluções viáveis $\{x | Ax \le b, x \in Z_+^n\}$ 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" [6, p. 259]

Exemplo 8.4 (Locação de facilidades não-capacitado)

$$
\begin{aligned}
\textbf{minimiza} \quad & \sum_{1\le j\le n} f_j y_j + \sum_{1\le i,j\le n} c_{ij} x_{ij} && && (8.1)\\
\textbf{sujeito a} \quad & \sum_{1\le j\le n} x_{ij} = 1, && \forall i = 1 \ldots n && (8.2)\\
& x_{ij} \le y_j, && \forall i,j = 1 \ldots n && (8.3)\\
& x_{ij} \in \mathbb{B}, && i,j = 1,\ldots,n && (8.4)\\
& y_j \in \mathbb{B}, && j = 1,\ldots,n. && (8.5)
\end{aligned}
$$

Ao invés de
$$x_{ij} \le y_j \tag{8.6}$$
podemos pensar em
$$\sum_{1\le i\le n} x_{ij} \le n y_j. \tag{8.7}$$

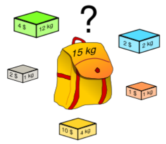
Essa formulação ainda é correto, mas usa n restrições ao invés de n^2. Entretanto, a qualidade da relação linear é diferente. É simples ver que podemos obter (8.7) somando (8.6) sobre todos i. Portanto, qualquer solução que satisfaz (8.6) satisfaz (8.7) também, e dizemos que (8.6) *domina* (8.7).
Que o contrário não é verdadeiro, podemos ver no seguinte exemplo: Com custos de instalação $f_j = 1$, de transporte $c_{ij} = 5$ para $i \ne j$ e $c_{ii} = 0$, duas cidades e uma fábrica obtemos as duas formulações (sem restrições de integralidade)

$$
\begin{array}{rll}
\textbf{minimiza} & y_1 + y_2 + 5c_{12} + 5c_{21} & y_1 + y_2 + 5c_{12} + 5c_{21}\\
\textbf{sujeito a} & x_{11} + x_{12} = 1 & x_{11} + x_{12} = 1\\
& x_{21} + x_{22} = 1 & x_{21} + x_{22} = 1\\
& y_1 + y_2 \le 1 & y_1 + y_2 \le 1\\
& x_{11} \le y_1 & x_{11} + x_{21} \le 2y_1\\
& x_{12} \le y_2 & \\
& x_{21} \le y_1 & x_{21} + x_{22} \le 2y_2\\
& x_{22} \le y_2 &
\end{array}
$$

A solução ótima da primeira 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!

$\diamond$

Exemplo: 0-1-**Knapsack**



$$\begin{aligned} \textbf{maximiza} \quad & \sum_{1\le i\le n} v_i x_i \\ \textbf{sujeito a} \quad & \sum_{1\le i\le n} p_i x_i \le P \\ & x_i \in \mathbb{B} \end{aligned}$$

Exemplo: $79x_1 + 53x_2 + 53x_3 + 45x_4 + 45x_5 \le 178$.

Exemplo: 0-1-**Knapsack**

- Observação: Para um subconjunto $S \subset [1, n]$: Se $\sum_S p_i > P$ então $\sum_S x_i \le |S| - 1$.
- Exemplos:

$$\begin{aligned} x_1 + x_2 + x_3 &\le 2 \\ x_1 + x_2 + x_4 + x_5 &\le 3 \\ x_1 + x_3 + x_4 + x_5 &\le 3 \\ x_2 + x_3 + x_4 + x_5 &\le 3 \end{aligned}$$

Exemplo: Casamento

- Dado um grafo $G = (V, A)$ procuramos um *casamento* máximo, i.e. um subconjunto $C \subseteq A$ tal que $\delta(v) \le 1$ para $v \in V$.

- Programa inteiro

$$\begin{aligned} \textbf{maximiza} \quad & \sum_A x_a \\ \textbf{sujeito a} \quad & \sum_{u\in N(v)} x_{(u,v)} \le 1, && \forall v \in V \\ & x_a \in \mathbb{B}, && \forall a \in A. \end{aligned}$$

Exemplo: Casamento

- Escolhe um subconjunto de nós $U \subseteq V$ arbitrário.
- Observação: O número de arestas internas é $\le \lfloor |U|/2 \rfloor$.
- Portanto:

$$\sum_{a\in U^2\cap A} x_a \le \lfloor |U|/2 \rfloor$$

é uma desigualdade válida.

Método de Chvátal-Gomory
Dado

$$\sum_i a_i x_i \le b$$

também temos, para $u \in \mathbb{R}, u > 0$ as restrições válidas

$$\begin{aligned} \sum_i u a_i x_i &\le ub && \text{(multiplicação)} \\ \sum_i \lfloor u a_i \rfloor x_i &\le ub && \lfloor y \rfloor \le y, 0 \le x_i \\ \sum_i \lfloor u a_i \rfloor x_i &\le \lfloor ub \rfloor && \text{Lado esquerda é inteira.} \end{aligned}$$

Método de Chvátal-Gomory

Teorema 8.2
Todas desigualdades válidadas pode ser construída através de um número finito de aplicações do método de Chvátal-Gomory.

Exemplo: Casamento

- Para um $U \subseteq V$ podemos somar as desigualdades

$$\sum_{u \in N(v)} x_{(u,v)} \leq 1 \qquad \forall v \in V$$

com peso 1/2, obtendo

$$\sum_{a \in U^2 \cap A} x_a + \frac{1}{2} \sum_{a \in N(U)} x_a \leq \frac{1}{2}|U|$$

- Também temos

$$\frac{1}{2} \sum_{a \in N(U)} x_a \geq 0$$

- Portanto

$$\sum_{a \in U^2 \cap A} x_a \leq \frac{1}{2}|U|$$
$$\sum_{a \in U^2 \cap A} x_a \leq \left\lfloor \frac{1}{2}|U| \right\rfloor \qquad \text{Lado esquerdo inteiro}$$

8.4 Planos de corte

Como usar restrições válidas?

- Adicionar à formulação antes de resolver.
  - Vantagens: Resolução com ferramentas padrão.
  - Desvantagens: Número de restrições pode ser 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

$$\max\{c^t x | Ax \leq b, x \in Z_+^n\}$$

- O que fazer, caso a relaxação linear não produz soluções ótimas?
- Um método: Introduzir *planos de corte*.

  Definição 8.3
  Um plano de corte (ingl. cutting plane) é uma restrição válida (ingl. valid inequality) que todas soluções inteiras satisfazem.

Algoritmo de planos de corte

PLANOS DE CORTE

Entrada Programa inteiro $\max\{c^t x | Ax \leq b, x \in Z_+^n\}$.

Saida Solução inteira ótima ou "Não existe corte.".

1 $V := \{x | Ax \leq b\}$ **{ região viável }**
2 $x^* := \text{argmax}\{c^t x | x \in V\}$ **{ resolve relaxação }**
3 **while** $(x^* \notin Z_+^n)$ **do**
4   **if** (existe corte $a^t x \leq d$ com $a^t x^* > d$) **then**
5     $V := V \cap \{x | a^t x \leq d\}$ **{ nova região viável }**
6     $x^* := \text{argmax}\{c^t x | x \in V\}$ **{ nova solução ótima }**
7   **else**
8   **return** "Não existe corte."
9   **end if**
10 **end while**

Método de Gomory

- Como achar um novo corte na linha 4 do algoritmo?
- 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

$$x_i = \bar{b}_i - \sum_{j\in\mathcal{N}} \bar{a}_{ij}x_j \qquad \text{Linha fracionária} \tag{8.8}$$

$$x_i \le \bar{b}_i - \sum_{j\in\mathcal{N}} \lfloor \bar{a}_{ij} \rfloor x_j \qquad \text{Definição de } \lfloor\cdot\rfloor \tag{8.9}$$

$$x_i \le \lfloor \bar{b}_i \rfloor - \sum_{j\in\mathcal{N}} \lfloor \bar{a}_{ij} \rfloor x_j \qquad \text{Integralidade de } x \tag{8.10}$$

$$0 \ge \{\bar{b}_i\} - \sum_{j\in\mathcal{N}} \{\bar{a}_{ij}\} x_j \qquad (8.8) - (8.10) \tag{8.11}$$

$$x_{n+1} = -\{\bar{b}_i\} + \sum_{j\in\mathcal{N}} \{\bar{a}_{ij}\} x_j \qquad \text{Nova variável} \tag{8.12}$$

$$x_{n+1} \in \mathbb{Z}_+ \tag{8.13}$$

(Para soluções inteiras, a diferença do lado esquerdo e do lado direito na equação (8.10) é inteira. Portanto x_{n+1} também é inteira.)
A solução básica atual não satisfaz (8.11), porque com $x_j = 0, j \in \mathcal{N}$ temos que satisfazer

$$\{\bar{b}_i\} \le 0,$$

uma contradição com a definição de $\{\cdot\}$ e o fato que $\bar{b}_i$ é fracionário. Portanto, provamos

Proposição 8.3
O corte (8.11) satisfaz os critérios da linha 4 do algoritmo PLANOS DE CORTE. Em particular, sempre existe um corte e o caso da linha 8 nunca se aplica.

Exemplo 8.5
Queremos resolver o problema

$$\begin{array}{rl} \textbf{maximiza} & x_1 + x_2 \\ \textbf{sujeito a} & -x_1 + 3x_2 \le 9 \\ & 10x_1 \le 27 \\ & x_1, x_2 \in \mathbb{Z}_+ \end{array}$$

A solução da relaxação linear produz a série de dicionários

$$\begin{array}{llllll} (1) & z & = & & x_1 & +x_2 \\ & w_1 & = 9 & & +x_1 & -3\mathbf{x_2} \\ & w_2 & = 27 & & -10x_1 & \end{array} \qquad \begin{array}{lllll} (2) & z & = 3 & +4/3x_1 & -1/3w_1 \\ & x_2 & = 3 & +1/3x_1 & -1/3w_1 \\ & w_2 & = 27 & -10\mathbf{x_1} & \end{array}$$

$$\begin{array}{lllll} (3) & z & = 6.6 & -4/30w_2 & -1/3w_1 \\ & x_2 & = 3.9 & -1/30w_2 & -1/3w_1 \\ & x_1 & = 2.7 & -1/10w_2 & \end{array}$$

A solução ótima $x_1 = 2.7$, $x_2 = 3.9$ é fracionária. Correspondendo com a segunda linha

$$x_2 \;= 3.9 \quad -1/30w_2 \quad -1/3w_1$$

temos o corte

$$w_3 \;= -0.9 \quad +1/30w_2 \quad +1/3w_1$$

e o novo sistema é

$$\begin{array}{lllll} (4) & z & = 6.6 & -4/30w_2 & -1/3w_1 \\ & x_2 & = 3.9 & -1/30w_2 & -1/3w_1 \\ & x_1 & = 2.7 & -1/10w_2 & \\ & w_3 & = -0.9 & +1/30w_2 & +1/3\mathbf{w_1} \end{array}$$

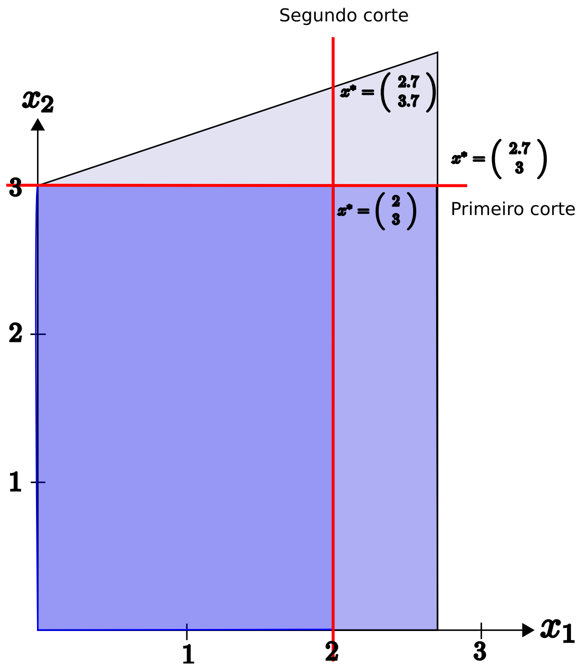
Esse sistema não é mais ótimo, e temos que re-otimizar. Pior, a solução básica atual não é viável! Mas como a na função objetivo todos coeficientes ainda são negativos, podemos aplicar o método Simplex dual. Um pivot dual gera a nova solução ótima

$$\begin{array}{lllll} (5) & z & = 5.7 & -1/10w_2 & -w_3 \\ & x_2 & = 3 & & -w_3 \\ & x_1 & = 2.7 & -1/10w_2 & \\ & w_1 & = 2.7 & -1/10w_2 & +3w_3 \end{array}$$

com $x_2 = 3$ inteiro agora, mas x_1 ainda fracionário. O próximo corte, que corresponde com x_1 é

$$\begin{array}{lllll} (5) & z & = 5.7 & -1/10w_2 & -w_3 \\ & x_2 & = 3 & & -w_3 \\ & x_1 & = 2.7 & -1/10w_2 & \\ & w_1 & = 2.7 & -1/10w_2 & +3w_3 \\ & w_4 & = -0.7 & +\mathbf{1/10w_2} & \end{array} \qquad \begin{array}{lllll} (6) & z & = 5 & -w_4 & -w_3 \\ & x_2 & = 3 & & -w_3 \\ & x_1 & = 2 & -w_4 & \\ & w_1 & = 2 & -w_4 & +3w_3 \\ & w_2 & = 7 & +10w_4 & \end{array}$$

cuja solução é inteira e ótima. $\Diamond$



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 *pura* 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.

8.5 Branch-and-bound

Branch-and-bound
Ramifica-e-limite (ingl. branch-and-bound)

- 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. [2]

- Idéia básica:
  - Particiona um problema recursivamente em subproblemas disjuntas e procura soluções.
  - Evite percorrer toda árvore de busca, calculando limites e cortando sub-árvores.
- Particularmente efetivo para programas inteiras: a relaxação linear fornece os limites.

Branch-and-bound

- Problema PI (puro): $\{\max c^t x | x \in S, x \in \mathbb{Z}_+^n\}$.
- Resolve a relaxação linear.
- Solução inteira? Problema resolvido.
- Senão: Escolhe uma variável inteira x_i, cuja solução atual v_i é fracionário.
- Tipico: Variável mais fracionário com $\text{argmin}_i |\{x_i\} - 0.5|$.
- Particione o problema $S = S_1 \dot\cup S_2$ tal que

$$S_1 = S \cap \{x | x_i \leq \lfloor v_i \rfloor\}; \quad S_2 = S \cap \{x | x_i \geq \lceil v_i \rceil\}$$

- Em particular com variáveis $x_i \in \mathbb{B}$:

$$S_1 = S \cap \{x | x_i = 0\}; \quad S_2 = S \cap \{x | x_i = 1\}$$

Limitar

- Para cada sub-árvore mantemos um limite inferior e um limite superior.
  - Limite inferior: Valor de uma solução encontrada na sub-árvore.
  - Limite superior: Valor da relaxação linear.
- Observação: A eficiencia do método depende crucialmente da qualidade do limite superior.
- Preferimos formulações mais “rigidos”.

Cortar sub-árvores

1. Corte por inviabilidade: Sub-problema é inviável.

2. Corte por limite: Limite superior da sub-árvore $\overline{z_i}$ menor que limite inferior global $\underline{z}$ (o valor da melhor solução encontrada).

3. Corte por otimalidade: Limite superior $\overline{z_i}$ igual limite inferior $\underline{z_i}$ da sub-árvore.

4. 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 pro profundidade
  - V: Limite superior encontrado mais rápido.
  - V: Pouco 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 superfluas.

- Busca por largura? Demanda de memória é impraticável.

Algoritmos B&B

B&B

Instância Programa inteiro $P = \max\{c^t x | Ax \leq b, x \in Z_+^n\}$.

Saida Solução inteira ótima.


```
{ usando função $\overline{z}$ para estimar limite superior }
$\underline{z}$:=$-\infty$          { limite inferior }
$A$:= $\{(P, g(P))\}$          { nós ativos }
while $A \neq \emptyset$ do
  Escolhe: $(P, g(P) \in A$;  $A := A \setminus (P, g(P))$
  Ramifique: Gera subproblemas $P_1, \ldots, P_n$.
  for all $P_i$, $1 \leq i \leq n$ do
    { adiciona, se permite melhor solução }
    if $\overline{z}(P_i) > \underline{z}$ then
      $A := A \cup \{(P_i, \overline{z}(P_i))\}$
    end if
    { atualize melhor solução }
    if (solução $\overline{z}(P_i)$ é viável) then
      $\underline{z} := \overline{z}(P_i)$
    end if
  end for
end while
```

9 Tópicos

Teorema 9.1 (Lenstra)
The integer programming feasibility problem can be solved with $O(p^{9p/2}L)$ arithmetic operations with integers of $O(p^{2p}L)$ bits in size, where p is the number of ILP variables and L is the number of bits in the input.

Observação: IPF is FPT.

Outras técnicas

- Branch-and-cut.

  Começa com menos restrições (relaxação) e insere restrições (cortes) nos sub-problemas da busca com branch-and-bound.

- Branch-and-price.

  Começa com menos variáveis e insere variáveis ("geração de colunas") nos sub-problemas da busca com branch-and-bound.

10 Exercícios

(Soluções a partir da página 185.)

Exercício 10.1 (Formulação)
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. Formalize como programa inteira.

Exercício 10.2 (Planos de corte)
Resolve

$$
\begin{array}{rl}
\textbf{maximiza} & x_1 + 3x_2 \\
\textbf{sujeito a} & -x_1 \leq -2 \\
& x_2 \leq 3 \\
& -x_1 - x_2 \leq -4 \\
& 3x_1 + x_2 \leq 12 \\
& x_i \in \mathbb{Z}_+
\end{array}
$$

e

$$
\begin{array}{rl}
\textbf{maximiza} & x_1 - 2x_2 \\
\textbf{sujeito a} & -11x_1 + 15x_2 \leq 60 \\
& 4x_1 + 3x_2 \leq 24 \\
& 10x_1 - 5x_2 \leq 49 \\
& x_1, x_2 \in \mathbb{Z}_+
\end{array}
$$

com o algoritmo de planos de corte using cortes de Chvátal-Gomory.

Exercício 10.3 (Formulação)
Para os problemas abaixo, acha 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 $|I|$.

CASAMENTO 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 *casamento 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 casamento.

PROBLEMA DE TRANSPORTE

Instância n depósitos, cada um com um estoque de p_i $(1 \le i \le n)$ produtos, e m clientes, cada um com uma demanda de d_j $(1 \le j \le m)$ produtos. Custos de transporte a_{ij} de cada depósito para cada cliente.

Solução Um decisão quantos produtos x_{ij} devem ser transportados do depósito i ao cliente j, 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 $\sum_{i,j} a_{ij} x_{ij}$.

CONJUNTO DOMINANTE

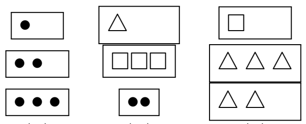
Instância Um grafo não-direcionado $G = (V, A)$.

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

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

Exercício 10.4 (Formulação: Apagando e ganhando)
Juliano é fã do programa de auditório Apagando e Ganhando, um programa no qual os participantes são selecionados atráves de um sorteio e recebem pr6emios 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 10.5 (Formulação: Set)
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) nào 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 10.6 (Matrizes totalmente unimodulares)
Para cada um dos problemas do exercício 10.3 decide, se a matriz de coeficientes é totalmento unimodular.

Exercício 10.7 (Formulação)
Para os problemas abaixo, acha uma formulação como programa inteira.

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 10.8 (Formulação e implementação: Investimento)
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 10.9 (Formulação e implementação: Brinquedos)
Um produtor de brinquedos projetou dois novos brinquedos para Natal. A preparação de uma fábrica para produzir custaria 50000 R$ para a primeiro brinquedo e 80000 R$ para o segundo. Após esse investimento inicial, o primeiro brinquedo rende 10 R$ por unidade e o segundo 15 R$.
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 e 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 e 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 deve ser produzido até Natal (incluindo o caso que um brinquedo não é produzido) de forma que maximiza o lucro total.

Exercício 10.10 (Formulação e implementação: aviões)
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 aviões, 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 produzidas	Cliente 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.

Parte III

Heurísticas

11 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: Criam soluções.
  - Heurísticas de busca: Procumra 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 a ótima.

Exemplo: Heurística construtiva

- Problema do Caixeiro Viajante (PCV) – Heurística do vizinho mais próximo.

HVizMaisProx

Entrada Matriz de distâncias completa $D = (d_{ij})$, número de cidades n.

Saída Uma solução factível do PCV: Ciclo Hamiltaneo C com custo c.

```
HVizMaisProx(D,n)=
  { cidade inicial randômica }
  u := seleciona uniformemente de [1,n]
  w := u
  { representação de camnihos: 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 i,a 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 nem 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

$$\begin{array}{rl} \textbf{optimiza} & f(x) \\ \textbf{sujeito a} & x \in S \end{array}$$

- S discreto: problema combinatorial.

Técnicas de solução

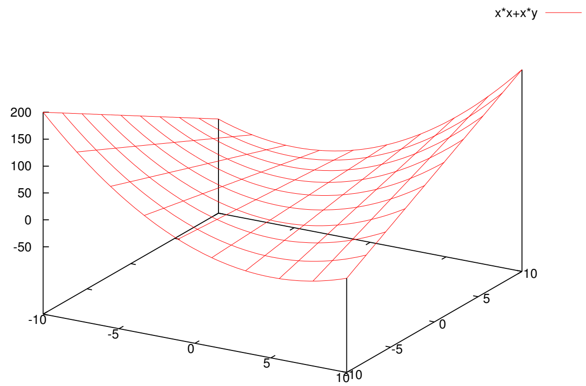
- Resolver o problema nessa geralidade: enumeração.
- Frequentemente: Uma solução $x \in S$ possiu uma *estrutura*.
- Exemplo: x é um tuplo, um grafo, etc.
- Permite uma enumeração por componente: branch-and-bound.

12 Heurísticas baseados em Busca local

12.1 Busca local

Busca Local

- Frequentemente: O espaço de soluções possui uma *topologia*.
- Exemplo da 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 TSP, qual o vizinho de um ciclo Hamiltaneo?
- Temos que definir uma vizinhança.

- Notação: Para $x \in S$
$$\mathcal{N}(x)$$
denota o conjunto de soluções vizinhos.
- 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

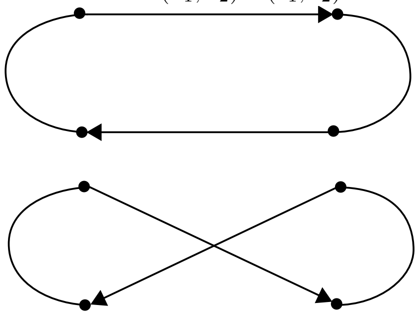
- k-change: mudança de k componentes da solução.
- Cada solução possui vizinhança de tamanho $O(n^k)$.
- Exemplo: 2-change, 3-change.
- TSP: 2-change (inversão).
- Inserção/remoção: inserção de um componente da solução, seguido da factibilização da solução
- Vertex cover: 1-change + remoção.

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-opt: 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-opt $|\mathcal{N}(s)| \in \mathrm{O}(n^k)$.

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*)

$$\begin{aligned} \forall x, y \in S\ \exists z_1, \ldots, z_k \in S \quad & z_1 \in \mathcal{N}(x) \\ & z_{i+1} \in \mathcal{N}(z_i) \qquad 1 \le i < k \\ & y \in \mathcal{N}(z_k) \end{aligned}$$

Busca Local: Idéia

- 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.

Busca local – Caso contínuo

BUSCA LOCAL CONTÍNUA

Entrada Solução inicial $s_0 \in \mathbb{R}^n$, tamanho incial α de um passo.

Saída Solução $s \in \mathbb{R}^n$ tal que $f(s) \le f(s_0)$.

Nome Gradient descent.

```
BuscaLocal(s0,α)=
  s := s0
  while ∇f(x) ≠ 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

Busca Local BI

Entrada Solução inicial s_0.

Saída Solução s tal que $f(s) \le f(s_0)$.

Nomes Steepest descent, steepest ascent.

BuscaLocal(s_0)=
  $s := s_0$
  while true
    $s' := \operatorname{argmin}_y \{f(y) \mid y \in \mathcal{N}(s)\}$
    if $f(s') < f(s)$ **then** $s := s'$
    else break
  end while
  return s

Busca Local – First Improvement

Busca Local FI (s)

Entrada Solução inicial s_0.

Saída Solução s' tal que $f(s') \le f(s)$.

Nomes Hill descent, hill climbing.

BuscaLocal(s_0)=
  $s := s_0$
  repeat
    Select any $s' \in \mathcal{N}(s)$ not yet considered
    if $f(s') < f(s)$ **then** $s := s'$
  until all solutions in $\mathcal{N}(s)$ have been visited
  return 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?

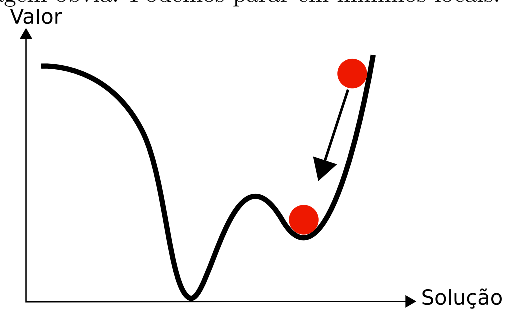
- Poucos resultados teóricos.
- Difícil de saber a qualidade da solução resultante.
- Depende de experimentos.

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 Simlated Annealing
  - Variable neighborhood search

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

MULTI-START

Entrada Número de repetições n.

Saída Solução s.

```
Multi_Start(n) :=
  s* := ∅
  f* := ∞
  repeat n vezes
    gera solução randômica s
    s := BuscaLocal(s)
    if f(s) < f* then
      s* := s
      f* := f(s)
    end if
  end repeat
  return s*
```

Cobrimento de Vértices

- Definição de vizinhança
- grafo sem vértices
- grafo estrela
- clique bipartido $K_{i,j}$
- grafo linha

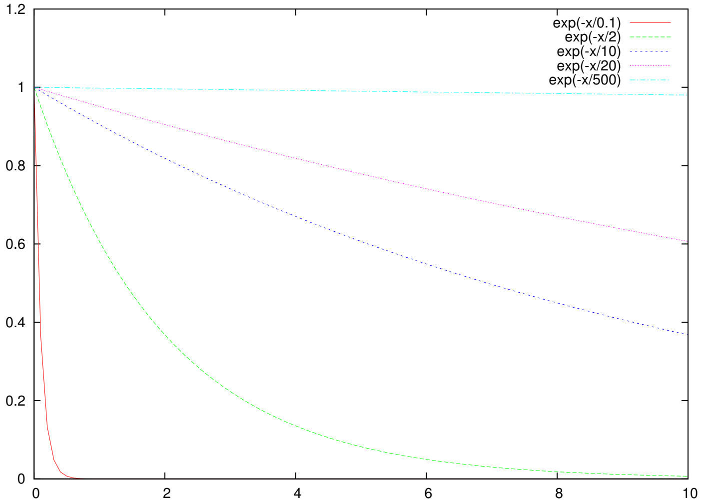
12.2 Metropolis e Simulated Annealing

O algoritmo Metropolis

- Proposto em 1953 por Metropolis, Rosenbluth, Rosenbluth, Teller e Teller
- 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 à função $e^{-\frac{E}{kT}}$ de Gibbs-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 em estado de baixa energia é maior que um 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
- Gere uma perturbação na solução s gerando uma solução s'.
- Se $E(s') \leq E(s)$ atualize a nova solução para s'.
- Caso contrário, $\triangle E = E(s') - E(s) > 0$.
- A solução s' passa ser a solução atual com probabilidade $e^{-\frac{\triangle E}{kT}}$
- Característica marcante: permite movimentos de melhora e, com baixa probabilidade, também de piora

Metropolis

METROPOLIS

Entrada Solução inicial s, uma temperatura T, uma constante k.

Saída Solução $s' : c(s') \leq c(s)$

```
Metropolis(s, T, k)=
  while STOP1 times do
    Select any unvisited s' ∈ N(s)
    if c(s') ≤ c(s) then update s := s'
    else
      with probability e^(-(c(s')-c(s))/kT) update s := s'
    end while
  return s
```

Considerações sobre o algoritmo

- O algoritmo Metropolis pode resolver problemas que o gradiente descendent não conseguia
- Mas em muitos casos o comportamento deste algoritmo não é desejado (vertex cover para grafo sem arcos)
- Alta probabilidade de saltos quando próximo de um mínimo local
- T pode ser manipulada: se T for alta, o algoritmo Metropolis funciona de forma similar a um *random walk* e se T for baixa (próxima a 0), o algoritmo Metropolis funciona de forma similar ao gradiente descendente.

Simulated Annealing

- Simula um processo de annealing.
- Annealing: 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
- Simulated annealing: parte de uma alta temperatura e baixa gradualmente. Para cada temperatura, permite um número máximo de saltos (dois loops encadeados)

Simulated Annealing

SIMULATED ANNEALING

Entrada Solução inicial s, temperatura T, constante k, fator de esfriamento $r \in [0,1]$, dois números inteiros STOP1, STOP2.

Saída Solução s' tal que $f(s') \leq f(s)$.

```
SimulatedAnnealing(s, T, k, r, STOP1, STOP2) :=
  repeat STOP2 vezes
    repeat STOP1 vezes
      seleciona s' ∈ N(s) que ainda não foi visitado
      if f(s') ≤ f(s) then
        s := s'
      else
        Com probabilidade e^{-(f(s')-f(s))/kT}: s := s'
      end fi
    end repeat
    T := T × r
  end repeat
return s
```

12.3 GRASP

GRASP

- GRASP: greedy randomized adaptive search procedure
- Proposto por Mauricio Resende e Thomas Feo (1989).
- Mauricio Resende: Pesquisador da AT&T por 20 anos, Departamento de Algoritmos e Otimização

Mauricio G. C. Resende

GRASP

- Método multi-start, em cada iteração
  1. Gera soluções com um procedimento guloso-randomizado.
  2. Otimiza as soluções geradas com busca local.

GRASP

Entrada Solução inicial s, parametro α.

Saída Solução $s' : c(s') \leq c(s)$

```
GRASP(s_0, α, ...)=
  s := s_0
  do
    s' := greedy_randomized_solution(α)
    s' := BuscalLocal(s')
    s := s' if f(s') < f(s)
  until a stopping criterion is satisfied
  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 $\alpha\%$ *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

GRASP

Entrada Solução inicial s, parametro α.

Saída Solução $s' : c(s') \le c(s)$

```
GRASP(s_0, α, ...) =
  x := s_0
  do
    y := greedy_randomized_solution(α)
    y := BuscalLocal(y)
    atualiza x caso y é solução melhor
  until a stopping criterion is satisfied
  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

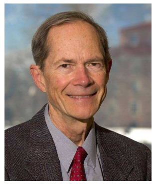
Comparação entre as metaheurísticas apresentadas

- 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

12.4 Busca Tabu

Busca Tabu (Tabu Search)

- Proposto por Fred Glover em 1986 (princípios básicos do método foram propostos por Glover ainda em 1977)
- Professor da Universidade do Colorado, EUA



Fred Glover

Busca Tabu (BT)

- Assim como em simulated anneling (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

BuscaTabu

Entrada uma solução s

Saída uma solução $s' : f(s') \le f(s)$

```
BuscaTabu()=
  Inicialização:
    s := S_0;  f* := f(s_0);  s* := s_0  ;  T := ∅
  while not STOP
```

```
  s' := select s' ∈ Ñ(s) com min f(s)
  if f(s) < f* then
    f* := f(s); s* := s
  insira movimento em T (a lista tabu)
end while
```

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: permiter 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

12.5 Variable Neighborhood Search

Variable Neighborhood Search

- Pierre Hansen e Mladenović, 1997
- Hansen é Professor na HEC Montréal, Canadá

Pierre Hansen

Variable Neighborhood Search

- Método multi-start que explora mais de uma vizinhaça.
- 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

Variable Neighborhood Search

VNS

Entrada Solução inicial s_0, um conjunto de vizinhanças $\mathcal{N}_i$, $1 \le i \le m$.

Saída uma solução $s : f(s) \le f(s_0)$

```
VNS(s0 ,{N_i})=
  x := s0
  do (até chegar a um mínimo local
      para todas as buscas locais)
    k := 1
    while k < m do
      escolhe y ∈ N_k(x) randomicamente
```



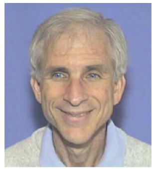
```
      y := BuscaLocal(y)
      if f(y) < f(x) then
        x := y
        k := 1
      else
        k := k+1
      end if
    end while
  end do
  return x
```

13 Heurísticas inspirados da natureza

13.1 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).



John Henry Holland (+1929)

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.

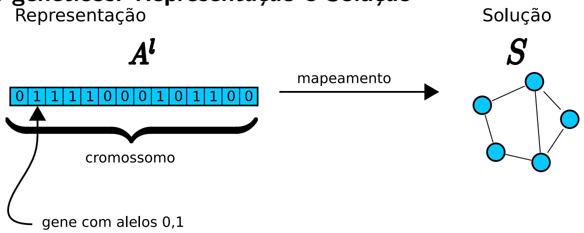
Algoritmos genéticos: Características

- 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

  Gens: 0 ou 1

  Cromossomo: 0001101010101011110110
- Problema do Caixeiro viajante

  Gens: 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

AlgoritmoGenético

Entrada Parâmetros do algoritmo.

Saída Melhor solução encontrada para o problema.

```
Inicialização e avalição 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 espaço de busca.
- Exemplo para representação binária: flip de k bits.
- Exemplo para o PCV: troca de posição entre duas cidades.

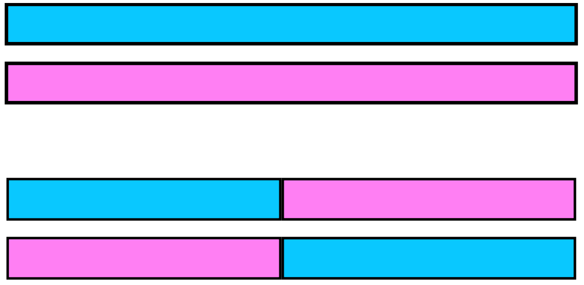
Recombinação

- 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.

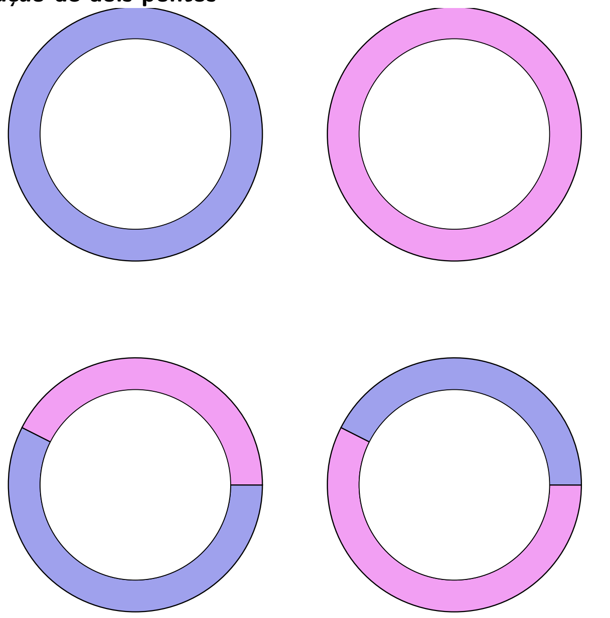
One-point crossover

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



Recombinação de dois pontos



Exemplo: Strategic Arc Crossover

- Selecione todos os pedaçõs 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 individuos 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.
- Idéia: Informação "cultural" pode ser adicionada a um indivíduo, gerando um algoritmo memético.
- Meme: unidade de informação cultural.

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 umum 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?

TBD

- Aplicação da programação linear na aproximação (TK 11.6,11.7, CA 2.4.1,2.4.2).
- ϵ-cortes no branch-and-bound: corta sub-árvores que são pouco mais que o valor atual.

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 $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, e para um dos conjuntos C acima, $C_+ := \{x \in C | x > 0\}$ e $C_- := \{x \in C | x < 0\}$. Por exemplo

$$\mathbb{R}_+ = \{x \in \mathbb{R} | x > 0\}.$$

Para um conjunto finito S, $\mathcal{P}(S)$ denota o conjunto de todos subconjuntos de S.

$A = (a_{ij}) \in F^{m \times n}$ denota uma *matriz* de m linhas e n colunas com elementos em F, a_i, com $a_i^t \in F^n$ a i-ésigma linha e $a^j \in F^m$ a j-ésima coluna de A.

- Vetores linearmente independentes.
- Poesto (linha,coluna) de uma matriz.

Definição A.1
Uma função $f : \mathbb{R} \to \mathbb{R}$ é *linear* se

1. $\forall a \in \mathbb{R}\ f(ax) = af(x)$
2. $f(x + y) = f(x) + f(y)$

Definição A.2 (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

$$\lfloor x \rfloor = \max\{y \in \mathbb{Z} | y \leq x\}$$
$$\lceil x \rceil = \min\{y \in \mathbb{Z} | y \geq x\}$$

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

$$x \leq \lceil x \rceil < x + 1 \qquad \text{(A.1)}$$
$$x - 1 < \lfloor x \rfloor \leq x \qquad \text{(A.2)}$$

B Formatos

Essa capítulo contém um breve resumo de dois formatos usados para descrever problemas de otimização linear. CPLEX LP é um formato simples, enquanto AMPL (A modeling language for mathematical programming) é uma linguagem completa para definir problemas de otimização, com elementos de programação, comandos interativos e um interface para diferentes "solvers" de problemas.
CPLEX LP serve bom para experimentos rápidos. Aprender AMPL precisa mais investimento, que rende em aplicações maiores. AMPL tem o apoio da maioria das ferramentas disponíveis.
Vários outros formatos são em uso, a maioria deles comerciais. Exemplos são MPS (Mathematical programming system, um formato antigo e pouco usável do IBM), LINGO, ILOG, GAMS e ZIMPL.

B.1 CPLEX LP

Uma gramática simplificada[1] 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*⟩

⟨*cmp*⟩ ::= '`<`' | '`<=`' | '`=`' | '`>`' | '`>=`'

[1] A gramática não contém as especificações "semi-continuous" e "SOS".

⟨*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 outras limites são necessárias, eles devem ser informados na seção "BOUNDS". A seções "GENERAL" e "BINARY" permitem restringir variáveis para $\mathbb{Z}$ e $\{0,1\}$, respectivamente.
As palvaras-chaves também podem ser escritem com letras minúscolas: o formato permite algumas abreviações não listadas acima (por exemplo, escrever "s.t" ao invés de "subject to").

Exemplo B.1
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
```

◊

B.2 AMPL

Objetos de modelagem

- Um modelo em AMPL consiste em
  - parâmetros,
  - variáveis,

– restrições, e
– objetiovos

- AMPL usa *conjuntos* (ou arrays de multiplas dimensões)

$$A : I \to D$$

mapeam 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 é um comando ou uma declaração.

- Parte de dados

```
data
<d1>
...
<dn>
end;
```

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

- Única 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 !=
- Pertinencia 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

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
- uma tabela

Exemplo B.2 (Exemplo 1.1 em AMPL)

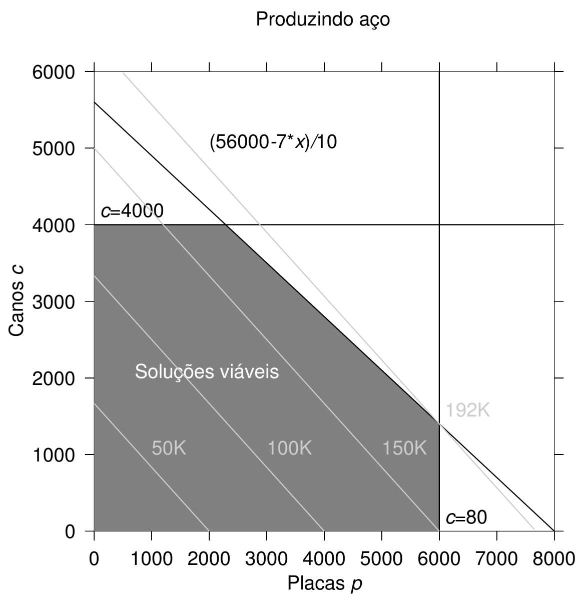
```
var c;  # número de croissants
var s;  # número de strudels
param lucro_croissant;      # o lucro por croissant
param lucro_strudel;    # o lucro por strudel
maximize lucro: lucro_croissant*c+lucro_strudel*s;
subject to ovo: c+1.5*s <= 150;
subject to acucar: 50*c+50*s <= 6000:
subject to croissant: c <= 80;
subject to strudel: s <= 60;
```

◊

C Soluções dos exercícios

Solução do exercício 5.6.

$$\begin{aligned}
\textbf{maximiza} \quad & 25p + 30c \\
\textbf{sujeito a} \quad & p/200 + c/140 \leq 40 \iff 7p + 10c \leq 56000 \\
& p \leq 6000 \\
& c \leq 4000 \\
& c, p \geq 0
\end{aligned}$$



A solução ótima é $p = 6000$, $c = 1400$ com valor 192000.

Solução do exercício 5.3.

$$\begin{aligned}
\textbf{maximiza} \quad & 2A + B \\
\textbf{sujeito a} \quad & A <= 6000 \\
& B <= 7000 \\
& A + B <= 10000
\end{aligned}$$

Resposta: A=6000 e B=4000 e Z=16000

Solução do exercício 5.5.
São necessárias cinco variáveis:

- x1: número de pratos de lasanha comidos por Marcio
- x2: número de pratos de sopa comidos por Marcio
- x3: número de pratos de hambúrgueres comidos por Renato
- x4: número de pratos de massa comidos por vini
- x5: números de pratos de sopa comidos por vini

Formulação:

$$\begin{aligned}
\textbf{maximiza} \quad & x1 + x2 + x3 + x4 + x5 \\
\textbf{sujeito a} \quad & 4 \geq x1 + x2 \geq 2 \\
& 5 \geq x3 \geq 2 \\
& 4 \geq x4 + x5 \geq 2 \\
& 70(x2 + x5) + 200x1 + 100x3 + 30x4 \leq 1000 \\
& 30(x2 + x5) + 100x1 + 100x3 + 100x4 \leq 800
\end{aligned}$$

Solução do exercício 5.7.
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

$$a_1 + b_1 + c_1 + a_3 + b_3 + c_3 \le 30$$
$$a_2 + b_2 + c_2 + a_3 + b_3 + c_3 \le 30$$

e os limites da predição

$$\begin{array}{lll} a_1 \le 4; & a_2 \le 8; & a_3 \le 3 \\ b_1 \le 8; & b_2 \le 13; & b_3 \le 10 \\ c_1 \le 22; & c_2 \le 20; & c_3 \le 18 \end{array}$$

Obviamente, todas variáveis também devem ser positivos.

Solução do exercício 5.8.

$$\begin{array}{rl} \textbf{maximiza} & z = 5x_1 + 5x_2 + 5x_3 \\ \textbf{sujeito a} & -6x_1 - 2x_2 - 9x_3 \le 0 \\ & -9x_1 - 3x_2 + 3x_3 \le 0 \\ & 9x_1 + 3x_2 - 3x_3 \le 0 \\ & x_j \ge 0 \end{array}$$

$$\begin{array}{rl} \textbf{maximiza} & z = -6x_1 - 2x_2 - 6x_3 + 4x_4 + 4x_5 \\ \textbf{sujeito a} & -3x_1 - 8x_2 - 6x_3 - 7x_4 - 5x_5 \le 3 \\ & 3x_1 + 8x_2 + 6x_3 + 7x_4 + 5x_5 \le -3 \\ & 5x_1 - 7x_2 + 7x_3 + 7x_4 - 6x_5 \le 6 \\ & x_1 - 9x_2 + 5x_3 + 7x_4 - 10x_5 \le -6 \\ & -x_1 + 9x_2 - 5x_3 - 7x_4 + 10x_5 \le 6 \\ & x_j \ge 0 \end{array}$$

$$\begin{array}{rl} \textbf{maximiza} & z = 7x_1 + 4x_2 + 8x_3 + 7x_4 - 9x_5 \\ \textbf{sujeito a} & -4x_1 - 1x_2 - 7x_3 - 8x_4 + 6x_5 \le -2 \\ & 4x_1 + x_2 + 7x_3 + 8x_4 - 6x_5 \le 2 \\ & -x_1 - 4x_2 - 2x_3 - 2x_4 + 7x_5 \le 7 \\ & -8x_1 + 2x_2 + 8x_3 - 6x_4 - 7x_5 \le -7 \\ & 8x_1 - 2x_2 - 8x_3 + 6x_4 + 7x_5 \le 7 \\ & x_j \ge 0 \end{array}$$

$$\begin{array}{rl} \textbf{maximiza} & z = 6x_1 - 5x_2 - 8x_3 - 7x_4 + 8x_5 \\ \textbf{sujeito a} & -5x_1 - 2x_2 + x_3 - 9x_4 - 7x_5 \le 9 \\ & 5x_1 + 2x_2 - x_3 + 9x_4 + 7x_5 \le -9 \\ & 7x_1 + 7x_2 + 5x_3 - 3x_4 + x_5 \le -8 \\ & -7x_1 - 7x_2 - 5x_3 + 3x_4 - x_5 \le 8 \\ & -5x_1 - 3x_2 - 5x_3 + 9x_4 + 8x_5 \le 0 \\ & x_j \ge 0 \end{array}$$

Solução do exercício 5.9.
Solução com método Simplex, escolhendo como variável entrante sempre aquela com o maior coeficiente positivo (em negrito):

$$\begin{array}{lllll} z & = & & 25p & +30c \\ \hline w_1 & = 56000 & & -7p & -10c \\ w_2 & = 6000 & & -p & \\ w_3 & = 4000 & & & -\mathbf{c} \end{array}$$

$$\begin{array}{llll} z & = 120000 & +25p & -30w_3 \\ \hline w_1 & = 16000 & -7\mathbf{p} & +10w_3 \\ w_2 & = 6000 & -p & \\ c & = 4000 & & -w_3 \end{array}$$

$$\begin{array}{llll} z & = 1240000/7 & -25/7p & +40/7w_3 \\ \hline p & = 16000/7 & -1/7w_1 & +10/7w_3 \\ w_2 & = 26000/7 & +1/7w_1 & -10/7\mathbf{w_3} \\ c & = 4000 & & -w_3 \end{array}$$

$$\begin{array}{llll} z & = 192000 & -3w_1 & -4w_2 \\ \hline p & = 6000 & & -w_2 \\ w_3 & = 2600 & +1/10w_1 & -7/10w_2 \\ c & = 1400 & -1/10w_1 & +7/10w_2 \end{array}$$

Solução do exercício 5.11.
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

$$\frac{2^2n}{n+1}\binom{2n}{n} \le \binom{2(n+1)}{n+1} \le 2^2\binom{2n}{n}.$$

Logo, por indução $(1/2n)2^{2n} \le \binom{2n}{n} \le 2^{2n}$.

Solução do exercício 10.2.
O sistema inicial

$$\begin{array}{rlll} z = & & x_1 & +3x_2 \\ w_1 = & -2 & +x_1 & \\ w_2 = & 3 & & -x_2 \\ w_3 = & -4 & +x_1 & +x_2 \\ w_4 = & 12 & -3x_1 & -x_2 \end{array}$$

não é primalmente nem dualmente viável. Aplicando a fase I (pivots x_0–w_3, x_0–x_1) e depois fase II (pivots x_2–w_1, w_3–w_2, w_1–w_4) gera o dicionário final

$$\begin{array}{rlll} z = & 12 & -8/3w_2 & -1/3w_4 \\ x_2 = & 3 & -w_2 & \\ w_3 = & 2 & -2/3w_2 & -1/3w_4 \\ x_1 = & 3 & +1/3w_2 & -1/3w_4 \\ w_1 = & 1 & +1/3w_2 & -1/3w_4 \end{array}$$

cuja solução $x_1 = 3$, $x_2 = 3$ já é inteira.
No segundo sistema começamos com o dicionário

$$\begin{array}{rlll} z = & & x_1 & -2x_2 \\ w_1 = & 60 & +11x_1 & -15x_2 \\ w_2 = & 24 & -4x_1 & -3x_2 \\ w_3 = & 59 & -10x_1 & +5x_2 \end{array}$$

e um pivot x_1–w_3 gera a solução ótimo fracionária

$$\begin{array}{rlll} z = & 4.9 & -0.1w_3 & -1.5x_2 \\ w_1 = & 113.9 & -1.1w_3 & -9.5x_2 \\ w_2 = & 4.4 & +0.4w_3 & -5x_2 \\ x_1 = & 4.9 & -0.1w_3 & +0.5x_2 \end{array}$$

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 pivot w_4–w_3 obtemos a solução ótima inteira

$$\begin{array}{rlll} z = & 4 & -w_4 & -x_2 \\ w_1 = & 104 & -11w_4 & -4x_2 \\ w_2 = & 8 & +4w_4 & -7x_2 \\ x_1 = & 4 & -w_4 & +1x_2 \\ w_3 = & 9 & +10w_4 & -5x_2 \end{array}$$

Solução do exercício 10.3.

Conjunto independente máximo Com variáveis indicadores x_v, $v \in V$ temos o programa inteiro

$$\begin{array}{lll} \textbf{maximiza} & \sum_{v\in V} x_v & \\ \textbf{sujeito a} & x_u + x_v \le 1, & \forall\{u,v\} \in A \qquad \text{(C.1)} \\ & x_v \in \mathbb{B}, & \forall v \in V. \end{array}$$

A equação C.1 garante que cada aresta possui no máximo um nó incidente.

Casamento perfeito com peso máximo Sejam x_a, $a \in A$ variáveis indicadores para a seleção de cada aresta. Com isso, obtemos o programa inteiro

$$\begin{array}{lll} \textbf{maximiza} & \sum_{a\in A} p(a)x_a & \\ \textbf{sujeito a} & \sum_{u\in N(v)} x_{\{u,v\}} = 1, & \forall v \in V \qquad \text{(C.2)} \\ & x_a \in \mathbb{B}, & \forall v \in V. \end{array}$$

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

$$\begin{aligned}
\textbf{minimiza} \quad & \sum_{\substack{1\le i\le n\\1\le j\le m}} c_{ij}x_{ij} & & \\
\textbf{sujeito a} \quad & \sum_{1\le j\le m} x_{ij} = p_i, & \forall 1\le i\le n \quad & \text{cada depósito manda todo estoque}\\
& \sum_{1\le i\le n} x_{ij} = d_j, & \forall 1\le j\le m \quad & \text{cada cliente recebe a sua demanda}\\
& x_{ij}\in\mathbb{Z}^+. & &
\end{aligned}$$

Conjunto dominante Sejam x_v, $v\in V$ variáveis indicadores para seleção de vértices. Temos o programa inteiro

$$\begin{aligned}
\textbf{minimiza} \quad & \sum_{v\in V} x_v & & \\
\textbf{sujeito a} \quad & x_v + \sum_{u\in N(v)} x_u \ge 1, & \forall v\in V \quad & \text{nó ou vizinho selecionado}\\
& x_v\in\mathbb{B}, & \forall v\in V. &
\end{aligned}$$

Solução do exercício 10.4.
Seja $d_1d_2\ldots d_n$ a entrada, e o objetivo selecionar $m\le n$ dígitos da entrada. Seja $x_{ij}\in\mathbb{B}$ um indicador que o dígito i da entrada seria selecionado como dígito j da saida, $1\le i\le n$, $1\le j\le m$. Então

$$\begin{aligned}
\textbf{maximiza} \quad & \sum_{i,j} x_{ij}d_i10^{m-j} & & \\
\textbf{sujeito a} \quad & \sum_{i} x_{ij} = 1, & \forall j \quad & \text{(C.3)}\\
& \sum_{j} x_{ij} \le 1, & \forall i \quad & \text{(C.4)}\\
& x_{ij} = 0, & \forall j > i, \quad & \text{(C.5)}\\
& x_{kl} \le 1 - x_{ij}, & \forall k > i, l < j. \quad & \text{(C.6)}
\end{aligned}$$

A função das equações é a seguinte:

- Equação C.3 garante que tem exatamente um dígito em cada posição.

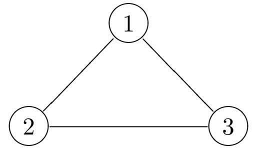
- Equação C.4 garante que cada dígito é selecionado no máximo uma vez.
- Equação C.5 garante que dígito i aparece somente a partir da posição j.
- Equação C.4 proibe inversões.

Solução do exercício 10.5.
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

$$\begin{aligned}
\textbf{maximiza} \quad & \sum_{1\le i\le 21} x_i\\
\textbf{sujeito a} \quad & Ax\le b\\
& x\ge 0.
\end{aligned}$$

Solução do exercício 10.6.

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 $0\le x_i\le 1$ é $x_1=x_2=x_3=1/2$ com valor $3/2$. (Observação: A transposta dessa matriz satisfaz os critérios (i) e (ii) 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).

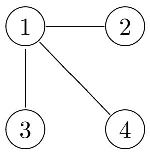
Casamento perfeito com peso máximo A matriz de coeficientes satisfaz critério (i). Ela tem uma linha para cada vértice e uma coluna para cada aresta do grafo. Como cada aresta é incidente a exatamente dois vértices, ela também satisfaz (ii). Finalmente, a bi-partição $V_1 \dot\cup V_2$ do grafo gera uma bi-partição das linhas que satisfaz (iii). Portanto, a matriz é TU, e o *Casamento 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

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

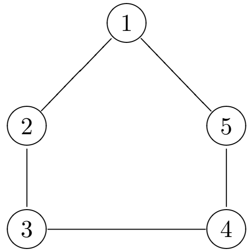
que *é* TU. Um contra-exemplo seria o grafo bi-partido $K_{1,3}$



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 ótimo $x_1 = x_2 = x_3 = x_4 = x_5 = 1/3$ com valor $5/3$ que não é inteira.

Solução do exercício 10.7.

Cobertura por arcos

$$\begin{aligned} \textbf{maximiza} \quad & \sum_{e \in E} c_e x_e \\ \textbf{sujeito a} \quad & \sum_{u \in N(v)} x_{uv} \geq 1, && \forall v \in V \\ & x_e \in \mathbb{B}. \end{aligned}$$

Observe que esse problema é redutível a um emparalhamento perfeito máximo e portanto possui solução em tempo polinomial.

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 $n = |V|$.

$$
\begin{aligned}
\textbf{minimiza} \quad & \sum_{1\le j\le n} c_j \\
\textbf{sujeito a} \quad & \sum_{1\le j\le n} x_{vj} = 1, && \forall v \in V && \text{(C.7)}\\
& x_{ui} + x_{vi} \le 1, && \forall \{u,v\} \in E, 1 \le i \le n && \text{(C.8)}\\
& nc_j \ge \sum_{v\in V} x_{vj}, && \forall 1 \le j \le n && \text{(C.9)}\\
& x_{vi}, c_j \in \mathbb{B}.
\end{aligned}
$$

- Equação C.7 garante que todo vértice recebe exatamente uma cor.
- Equação C.8 garante que vértices adjacentes recebem cores diferentes.
- Equação C.9 garante que $c_j = 1$ caso cor j for usada.

Clique mínimo ponderado

$$
\begin{aligned}
\textbf{minimiza} \quad & \sum_{v\in V} c_v x_v \\
\textbf{sujeito a} \quad & x_u + x_v \le 1, && \forall \{u,v\} \notin E && \text{(C.10)}\\
& x_v \in \mathbb{B}.
\end{aligned}
$$

Equação C.10 garante que não existe um par de vértices selecionados que não são vizinhos.

Subgrafo cúbico x_e indica se o arco e é selecionado, e y_e indica se ele possui grau 0 (caso contrário grau 3).

$$
\begin{aligned}
\textbf{minimiza} \quad & \sum_{e\in E} x_e \\
\textbf{sujeito a} \quad & \sum_{e\in N(v)} x_e \le 0 + |E|(1-y_e) \\
& \sum_{e\in N(v)} x_e \le 3 + |E|y_e \\
& -\sum_{e\in N(v)} x_e \le -3 + 3y_e
\end{aligned}
$$

Observe que o grau de cada vértice é limitado por $|E|$.

Solução do exercício 10.8.
Sejam $x_i \in \mathbb{B}, 1 \le i \le 7$ variáveis que definem a escolha do projeto i. Então temos

$$
\begin{aligned}
\textbf{maximiza} \quad & 17x_1 + 10x_2 + 15x_3 \\
& + 19x_4 + 7x_5 + 13x_6 + 9x_7 \\
\textbf{sujeito a} \quad & 43x_1 + 28x_2 + 34x_3 + 48x_4 \\
& + 17x_5 + 32x_6 + 23x_7 \le 100 && \text{Limite do capital}\\
& x_1 + x_2 \le 1 && \text{Projetos 1,2 mutualmente exclusivos}\\
& x_3 + x_4 \le 1 && \text{Projetos 3,4 mutualmente exclusivos}\\
& x_3 + x_4 \le x_1 + x_2 && \text{Projeto 3 ou 4 somente se 1 ou 2}
\end{aligned}
$$

http://www.inf.ufrgs.br/~mrpritt/e6q2.mod

```
set projetos := 1 .. 7;
param lucro { projetos };
param custo { projetos };

var fazer { projetos } binary;

maximize M: sum { i in projetos } lucro[i]*fazer[i];
subject to S1:
  sum { i in projetos } custo[i]*fazer[i] <= 100;
subject to S2: fazer[1]+fazer[2] <= 1;
subject to S3: fazer[3]+fazer[4] <= 1;
subject to S4: fazer[3]+fazer[4] <= fazer[1]+fazer[2];

data;
param lucro :=  1 17 2 10 3 15 4 19 5 7  6 13 7 9;
param custo :=  1 43 2 28 3 34 4 48 5 17 6 32 7 23;
end;
```

Solução: Selecionar projetos 1,3,7 com lucro de 41MR$.

Solução do exercício 10.9.
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 $1 \leq i \leq 2$).

$$
\begin{array}{lll}
\textbf{maximiza} & 10u_1 + 15u_2 - 50000b_1 - 80000b_2 & \\
\textbf{sujeito a} & u_i \leq Mb_i & \text{Permitir unidades somente se tem produção} \\
 & u_1/50 + u_2/40 \leq 500 + fM & \text{Limite fábrica 1, se selecionada} \\
 & u_1/40 + u_2/25 \leq 700 + (1-f)M & \text{Limite fábrica 2, se selecionada}
\end{array}
$$

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

```
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 10.10.
Sejam $a_i \in \mathbb{B}$ uma variável que determina se avião i vai ser produzido e $u_i \in \mathbb{Z}$ as unidadas produzidas.

$$
\begin{array}{lll}
\textbf{maximiza} & 2u_1 + 3u_2 + 0.2u_3 - 3a_1 - 2a_2 & \\
\textbf{sujeito a} & 0.2u_1 + 0.4u_3 + 0.2u_3 \leq 1 & \text{Limite de capacidade} \\
 & u_i \leq 5b_i & \text{Permitir unidades somente se for produzido} \\
 & u_1 \leq 3 & \text{Limite avião 1} \\
 & u_2 \leq 2 & \text{Limite avião 2} \\
 & u_3 \leq 5 & \text{Limite avião 3}
\end{array}
$$

http://www.inf.ufrgs.br/~mrpritt/e6q4.mod

```
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$.

Bibliografia

[1] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation – Combinatorial Optimization Problems and their Approximability Properties.* Springer-Verlag, 1999. INF 510.5 C737.

[2] J. Clausen. Branch and bound algorithms – principles and examples, 1999.

[3] N. Maculan and M. H. C. Fampa. *Otimização linear.* Editora UnB, 2006. INF 65.012.122 M175o.

[4] R. J. Vanderbei. *Linear programming: Foundations and Extensions.* Kluwer, 2nd edition, 2001.

[5] H. P. Williams. Fourier's method of linear programming and its dual. *The American Mathematical Monthly*, 93(9):681–695, 1986.

[6] L. A. Wolsey and G. L. Nemhauser. *Integer and Combinatorial Optimization.* Wiley, 1999.

Índice