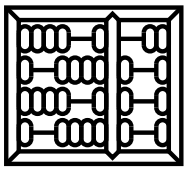


UNIVERSIDADE DE CAMPINAS - UNICAMP
INSTITUTO DE COMPUTAÇÃO - IC

Introdução a Programação Linear e Inteira

Flávio Keidi Miyazawa

Campinas, 2002-2016

Programação Linear

Programação linear

- dá a resolução exata para muitos problemas
- dá a resolução aproximada para muitos problemas
- faz parte dos principais métodos para obter soluções ótimas
- faz parte dos principais métodos para obter soluções aproximadas
- dá excelentes delimitantes para soluções ótimas
- pode ser executada muito rapidamente
- há diversos programas livres e comerciais

Obs.: Algumas teorias sobre a estrutura dos programas lineares serão apresentadas para se entender os resultados das reduções, mas o foco da aula será nas reduções para Programas Lineares.

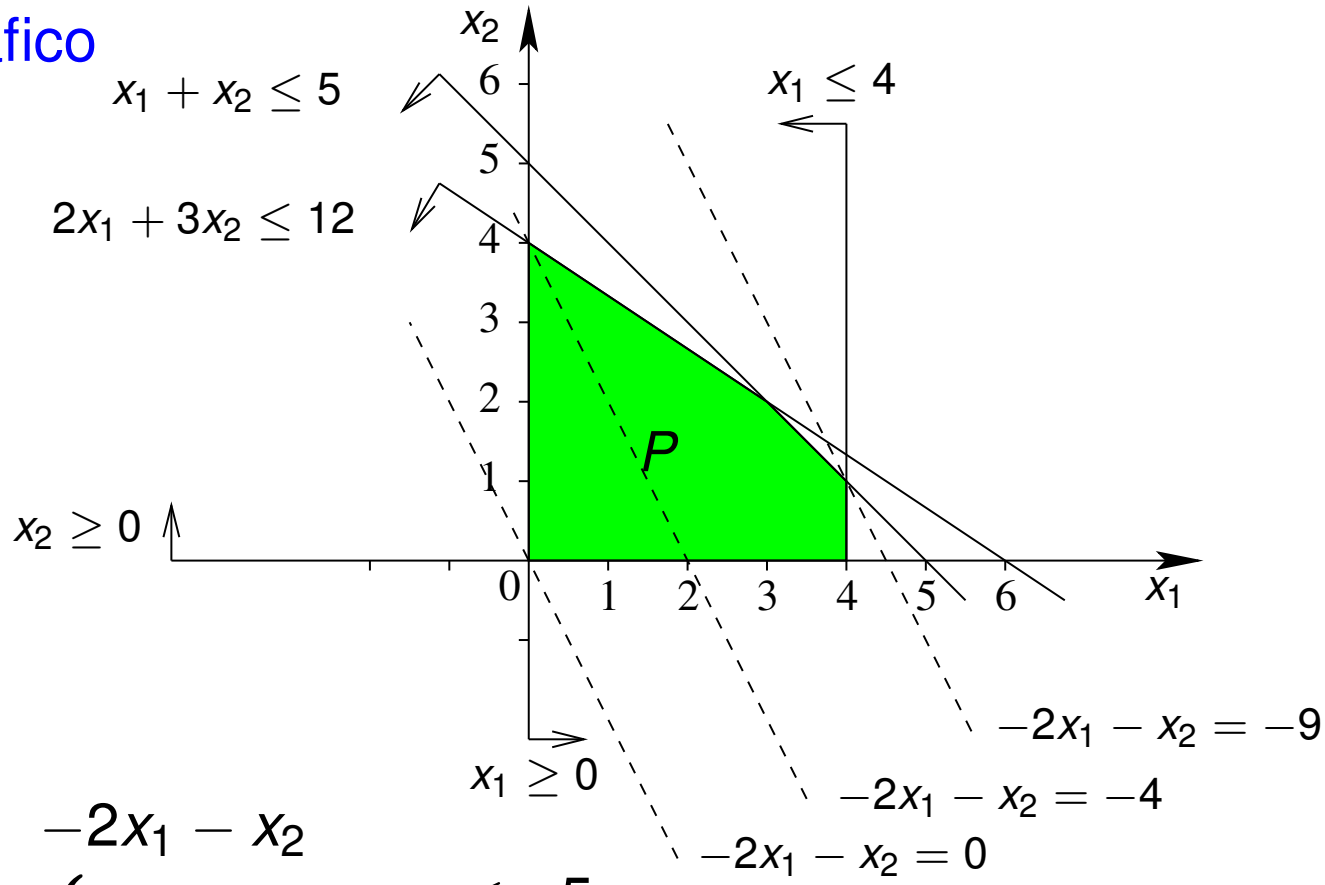
Programação Linear

Problema PL: Dados matriz $A = (a_{ij}) \in \mathbb{Q}^{n \times m}$, vetores $c = (c_i) \in \mathbb{Q}^n$ e $b = (b_i) \in \mathbb{Q}^m$, encontrar vetor $x = (x_i) \in \mathbb{Q}^m$ (se existir) que

$$\begin{array}{ll} \text{minimize} & c_1 x_1 + c_2 x_2 + \cdots + c_m x_m \\ \text{sujeito a} & \left\{ \begin{array}{lcl} a_{11} x_1 + a_{12} x_2 + \cdots + a_{1m} x_n & \leq & b_1 \\ a_{21} x_1 + a_{22} x_2 + \cdots + a_{2m} x_n & \geq & b_2 \\ \vdots & \vdots & \vdots \\ a_{n1} x_1 + a_{n2} x_2 + \cdots + a_{nm} x_n & = & b_n \\ x_i \in \mathbb{Q} & & \end{array} \right. \end{array}$$

Teorema: *PL pode ser resolvido em tempo polinomial.*

Exemplo gráfico



$$
\begin{array}{lll}
\text{minimize} & -2x_1 - x_2 & \\
\text{sujeito a} & \left\{\begin{array}{llll}
x_1 & +x_2 & \leq & 5 \\
2x_1 & +3x_2 & \leq & 12 \\
x_1 & & \leq & 4 \\
x_1 & & \geq & 0 \\
& x_2 & \geq & 0
\end{array}\right. &
\end{array}
$$

Sol. ótima: $x_1 = 4$ e $x_2 = 1$

Valor da sol. ótima = -9

Teorema: *Uma solução que é vértice do PL pode ser encontrada em tempo polinomial.*

Algoritmos polinomiais para resolver PL:

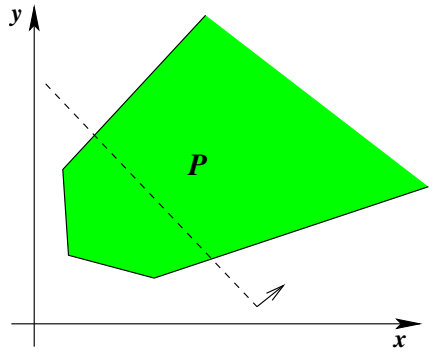
- Algoritmo dos elipsóides (Khachiyan'79) e
- Método dos pontos interiores (Karmarkar'84).

Algoritmos exponenciais para resolver PL:

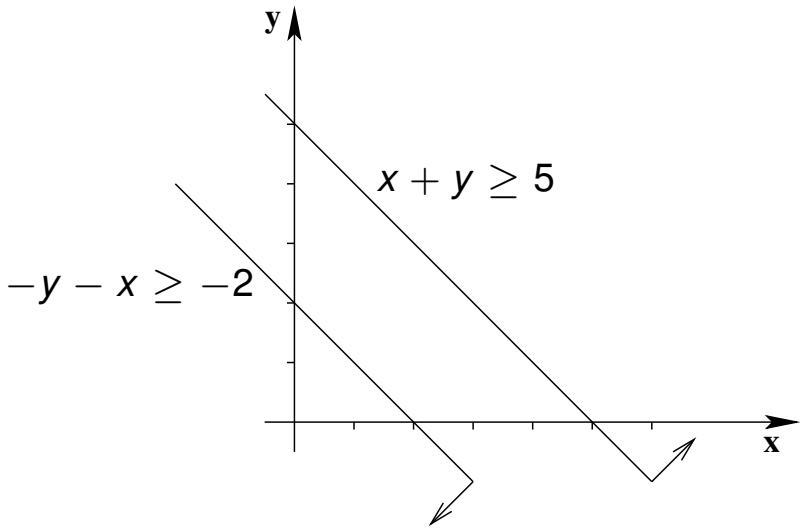
- Método simplex (Dantzig'47) (tempo médio polinomial).

Nem sempre encontramos uma solução ótima

- Sistema ilimitado



- Sistema inviável



Resolvedores de Sistemas Lineares

Programas comerciais

- CPLEX / IBM: `www.ilog.com/products/cplex/`
- XPRESS: `www.dashoptimization.com/`
- GUROBI:: `http://www.gurobi.com/`

Programas livres

- SCIP: `http://scip.zib.de/`
- COIN-CLP: `www.coin-or.org/Clp/`
- SoPlex / Zib: `www.zib.de/Optimization/Software/Soplex`
- GLPK / Gnu: `www.gnu.org/software/glpk/glpk.html`
- LEMON / Coin-OR: `http://www.coin-or.org/`
  Biblioteca de grafos usando solver GLPK ou COIN-CLP

Trecho de implementação usando LEMON

```
#include <lemon/lp.h> // g++ ex_lp1.cpp -lemon -lglpk -o ex_lp1
using namespace lemon;
using namespace std;
int main()
{ Lp lp;                    / Declarando/criando um LP (inicialmente v
  Lp::Col x1 = lp.addCol();  // Adicionando duas variáveis
  Lp::Col x2 = lp.addCol();
  lp.addRow(   x1 +    x2 <= 5 );   // Adicionando restrições
  lp.addRow( 2*x1 + 3*x2  <= 12);
  // Definindo os limitantes superior e inferior de cada variável
  lp.colLowerBound( x1, 0 );   lp.colUpperBound( x1, 4 );
  lp.colLowerBound( x2, 0 );
  // Definindo a função objetivo e se de minimização ou maximizaçã
  lp.obj( -2*x1 - x2 );   lp.min();
  lp.solve();         // Resolvendo o sistema
  if (lp.primalType() == Lp::OPTIMAL) {// Imprimindo valor das var
    cout << "Valor da funcao objetivo: " << lp.primal() << endl;
    cout << "x1 = " << lp.primal(x1) << endl;
    cout << "x2 = " << lp.primal(x2) << endl;
  } else {cout << "Nao encontrou solucao otima." << endl;}
  return 0;}
```

Problema da Dieta

São dados:

- m alimentos: $ali_1, \ldots, ali_m$
- n nutrientes: $nut_1, \ldots, nut_n$
- preço p_j de cada alimento ali_j
- recomendações diárias r_i de cada nutriente nut_i, para uma dieta balanceada
- quantidade q_{ij} do nutriente nut_i no alimento ali_j,

Objetivo: Encontrar uma dieta mais barata respeitando as recomendações nutricionais

Exemplo: Dieta I

Nutrientes, com recomendações diárias mínima e máxima

Nutriente	*mínimo*	*máxima*
nut_1 = *cálcio*	*800*	*1200*
nut_2 = *ferro*	*10*	*18*

Alimentos, com preço e composição nutricional:

Alimento	*preço*	*cálcio*	*ferro*
ali_1 = *carne de boi (kg)*	*20.0*	*110.00*	*29.00*
ali_2 = *feijão cozido (kg)*	*6.0*	*170.00*	*15.00*
ali_3 = *leite desnatado (lt)*	*2.0*	*1150.00*	*0.00*

Formulação Linear

Variáveis

- x_1 quantidade de *carne de boi* a comprar
- x_2 quantidade de *feijão cozido* a comprar e
- x_3 quantidade de *leite desnatado* a comprar.

Função objetivo

- Minimizar o preço pago por todos os alimentos. I.e.,

$$\text{minimize } 20{,}0x_1 + 6{,}0x_2 + 2{,}0x_3$$

Restrições

- quantidades x_1, x_2 e x_3 devem ser valores válidos e satisfazer restrições nutricionais

Restrições nutricionais

- Quantidade de cálcio consumido dentro das recomendações diárias

$$\begin{aligned} 110x_1 + 170x_2 + 1150x_3 &\geq 800 \\ 110x_1 + 170x_2 + 1150x_3 &\leq 1200 \end{aligned}$$

- Quantidade de ferro consumido dentro das recomendações diárias

$$\begin{aligned} 29x_1 + 15x_2 &\geq 10 \\ 29x_1 + 15x_2 &\leq 18 \end{aligned}$$

Restrições de não negatividade

- Nenhuma quantidade pode ser negativa

$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0$$

Formulação Linear

$$
\begin{array}{lll}
\text{minimize} & 20{,}0x_1 + 6{,}0x_2 + 2{,}0x_3 & \\
\text{sujeito a} & \left\{\begin{array}{rrrcl}
110x_1 + & 170x_2 + & 1150x_3 & \geq & 800 \\
110x_1 + & 170x_2 + & 1150x_3 & \leq & 1200 \\
29x_1 + & 15x_2 & & \geq & 10 \\
29x_1 + & 15x_2 & & \leq & 18 \\
x_1 \geq 0 & x_2 \geq 0 & x_3 \geq 0 & &
\end{array}\right.
\end{array}
$$

Resolvendo-se:

Obtemos uma dieta de 5,194 reais com

$x_1 = 0$ kg. de carne de boi,

$x_2 = 0.666$ kg. de feijão cozido e

$x_3 = 0.597$ lt. de leite desnatado.

A quantidade diária de cálcio nesta dieta é 800 e de ferro é 10.

Trecho de implementação usando LEMON

```
int main() {
 Lp lp;  // Declarando um LP (inicialmente vazio)
 typedef Lp::Col LPvar;
 // Adicionando três variáveis
 LPvar x1=lp.addCol(); LPvar x2=lp.addCol(); LPvar x3=lp.addCol();
 // Definindo a função objetivo e direção de minimização
 lp.obj( 20.0*x1 + 6.0*x2 + 2.0*x3 );   lp.min();
 lp.addRow( 800 <= 110*x1 + 170*x2 + 1150*x3 <= 1200 );
 lp.addRow(  10 <=  29*x1 +  15*x2           <= 18 );
 lp.colLowerBound( x1, 0 );    // restrição x1 >= 0
 lp.colLowerBound( x2, 0 );    // restrição x2 >= 0
 lp.colLowerBound( x3, 0 );    // restrição x3 >= 0
 lp.solve();   // Resolvendo o sistema
 if (lp.primalType() == Lp::OPTIMAL) {  // Imprimindo o valor das
   cout << "Valor da funcao objetivo: " << lp.primal() << endl;
   cout << "x1 = " << lp.primal(x1) << endl;
   cout << "x2 = " << lp.primal(x2) << endl;
   cout << "x3 = " << lp.primal(x3) << endl;
 } else {cout << "Nao encontrou solucao otima." << endl;}
 return 0;
```

Exemplo: Dieta II

Nutrientes, com recomendações diárias mínima e máxima

Nutriente	*mínimo*	*máxima*
nut_1 = *fósforo*	800	1200
nut_2 = *vitamina C*	60	90

Alimentos, com preço e composição nutricional:

Alimento	*preço*	*fósforo*	*vitamina C*
ali_1 = *carne de boi (kg)*	20.0	1800.00	0.00
ali_2 = *feijão cozido (kg)*	6.0	490.00	10.00

Variáveis

- x_1 quantidade de *carne de boi* a comprar
- x_2 quantidade de *feijão cozido* a comprar

Formulação Linear

$$
\begin{array}{lll}
\text{minimize} & 20{,}0x_1 + 6{,}0x_2 & \\
\text{sujeito a} & \left\{\begin{array}{rcl}
1800x_1 + 490x_2 & \geq & 800 \\
1800x_1 + 490x_2 & \leq & 1200 \\
10x_2 & \geq & 60 \\
10x_2 & \leq & 90 \\
x_1 \geq 0 \quad x_2 \geq 0 &&
\end{array}\right.
\end{array}
$$

Este sistema é inviável. De fato:

- O único a ter vitamina C é o feijão (e bem pouco)
- Por $10x_2 \geq 60$ temos que $x_2 \geq 6$
  I.e., para suprir necessidade de vitamina C, precisamos ingerir pelo menos 6 kg. de feijão por dia!!
- 6kg. de feijão contém $6 \star 490 = 2940$mg de fósforo
  acima do limite diário permitido de 1200mg de fósforo.

Otimização de Portfolio

- Temos 100.000 reais para investir em ações
- As ações selecionadas e a porcentagem de retorno esperado em 1 ano são:

Empresa	Retorno (em %)
emp_1 = Petrobrás (petróleo/estatal)	9.0%
emp_2 = Vale do Rio Doce (siderurgia)	10.2%
emp_3 = Votorantim (siderurgia)	6.5%
emp_4 = Texaco (petróleo)	9.5%
emp_5 = Sanasa (água/estatal)	8.5%

A recomendação dos especialistas é a seguinte:

- Recomenda se investir pelo menos 25% e no máximo 55% em empresas estatais.
- Petrobrás e Texaco são empresas do mesmo setor (petróleo). Recomenda-se que o investimento nas duas não passe de 55 %.
- Vale do Rio Doce e Votorantim são do mesmo setor (siderurgia) recomenda que o investimento nas duas não passe de 45 %.
- Apesar da Vale do Rio Doce ter a maior taxa de retorno, há boatos que ela pode estar maquiando faturamento. Recomenda-se que a quantidade de investimento nela não passe de 60% do total de investimento feito em empresas de siderurgia.

Formulação Linear

Variáveis

- x_1 quantidade de investimento na *Petrobrás*
- x_2 quantidade de investimento na *Vale do Rio Doce*
- x_3 quantidade de investimento na *Votorantim*
- x_4 quantidade de investimento na *Texaco*
- x_5 quantidade de investimento na *Sanasa*

Função objetivo

- Maximizar o lucro esperado,

$$\text{maximize } 0{,}090x_1 + 0{,}102x_2 + 0{,}065x_3 + 0{,}095x_4 + 0{,}085x_5$$

Restrições

- quantidades $x_1, \ldots, x_5$ devem ser valores válidos e devem satisfazer recomendações dos especialistas

Restrições impostas por especialistas

- Recomenda se investir pelo menos 25% e no máximo 55% em empresas estatais.

$$\begin{aligned} x_1 + x_5 &\geq 25000 \\ x_1 + x_5 &\leq 55000 \end{aligned}$$

- Petrobrás e Texaco são empresas do mesmo setor (petróleo). Recomenda-se que o investimento nas duas não passe de 55 %.

$$x_1 + x_4 \leq 55000$$

- Vale do Rio Doce e Votorantim são do mesmo setor (siderurgia) recomenda que o investimento nas duas não passe de 45 %.

$$x_2 + x_3 \leq 45000$$

- Apesar da Vale do Rio Doce ter a maior taxa de retorno, há boatos que ela pode estar maquiando faturamento. Recomenda-se que a quantidade de investimento nela não passe de 60% do total de investimento feito em empresas de siderurgia.

$$x_2 \leq 0.6(x_2 + x_3)$$

$$\text{I.e.,} \quad -0.4x_2 + 0.6x_3 \geq 0$$

Demais Restrições

- Total investido é 100.000

$$x_1 + x_2 + x_3 + x_4 + x_5 = 100000$$

- Nenhuma quantidade pode ser negativa

$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0, \quad x_4 \geq 0, \quad x_5 \geq 0$$

Formulação Linear

$$
\begin{array}{lll}
\text{maximize} & 0{,}090x_1 + 0{,}102x_2 + 0{,}065x_3 + 0{,}095x_4 + 0{,}085x_5 & \\
\text{sujeito a} & \left\{\begin{array}{rcl}
x_1 + x_2 + x_3 + x_4 + x_5 & = & 100000 \\
x_1 + x_5 & \geq & 25000 \\
x_1 + x_5 & \leq & 55000 \\
x_1 + x_4 & \leq & 55000 \\
x_2 + x_3 & \leq & 45000 \\
-.4x_2 + .6x_3 & \geq & 0 \\
x_1 \geq 0 \quad x_2 \geq 0 \quad x_3 \geq 0 \quad x_4 \geq 0 \quad x_5 \geq 0 &&
\end{array}\right.
\end{array}
$$

Resolvendo-se:

Obtemos uma lucro estimado de 9094 reais investindo

$x_1 = 0$ na Petrobrás, $x_2 = 12000$ na Vale do Rio Doce,

$x_3 = 8000$ na Votorantim, $x_4 = 55000$ na Texaco e

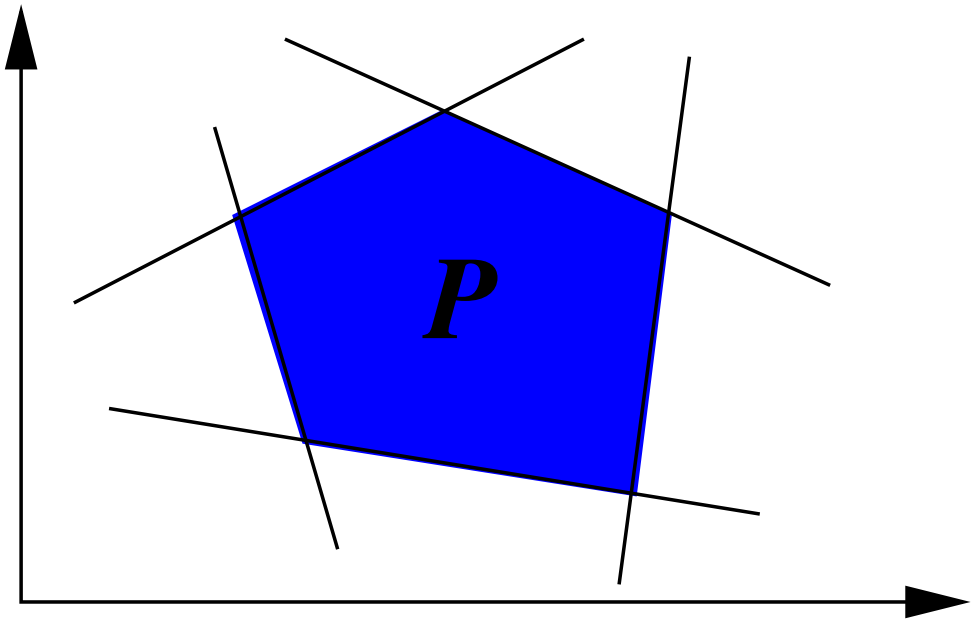
$x_5 = 25000$ na Sanasa.

Algumas Definições

Def.: Chamamos o conjunto de pontos P,

$$P := \left\{ x \in \mathbb{Q}^n : \begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_n & \leq & b_1 \\ \vdots & \vdots & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_n & \geq & b_n \end{array} \right\}$$

como sendo um poliedro.



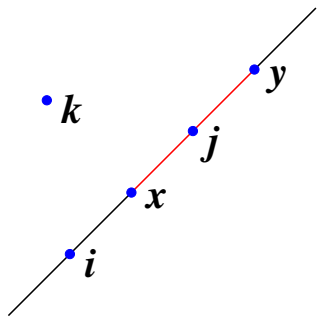
Def.: Dado conjunto de pontos S, dizemos que y é uma combinação convexa dos pontos de S se $y = \alpha_1 y_1 + \alpha_2 y_2 + \cdots + \alpha_n y_n$, onde $y_i \in S$, $\alpha_i \geq 0$ e $\sum_i \alpha_i = 1$

Exemplo: *Os pontos que são combinação convexa de dois pontos x e y são:*

$$\text{conv}(\{x, y\}) := \{\alpha_1 x + \alpha_2 y : \ \alpha_1 \geq 0, \ \alpha_2 \geq 0, \ \alpha_1 + \alpha_2 = 1\}$$

substituindo $\alpha_2 = 1 - \alpha_1$, temos

$$\text{conv}(\{x, y\}) := \{\alpha x + (1 - \alpha) y : \ 0 \leq \alpha \leq 1\}$$



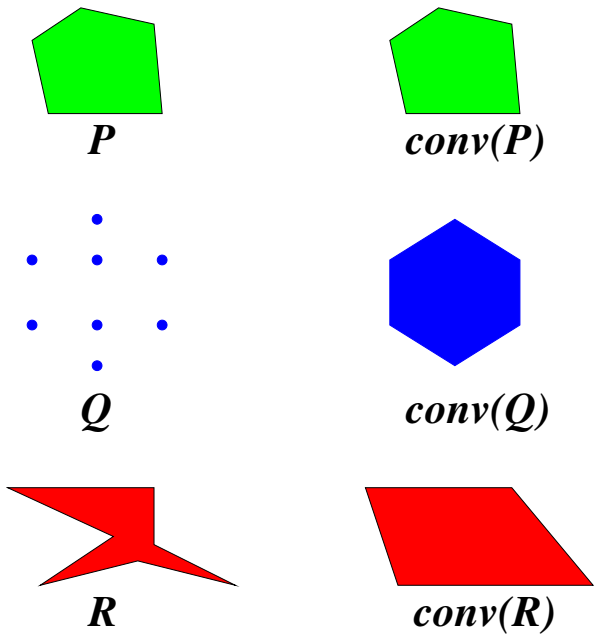
j é combinação convexa de x e y, mas i e k não são.

Exercício: Mostre que se $x, y \in conv(S)$ e $z \in conv(x, y)$, então temos também que $z \in conv(S)$.

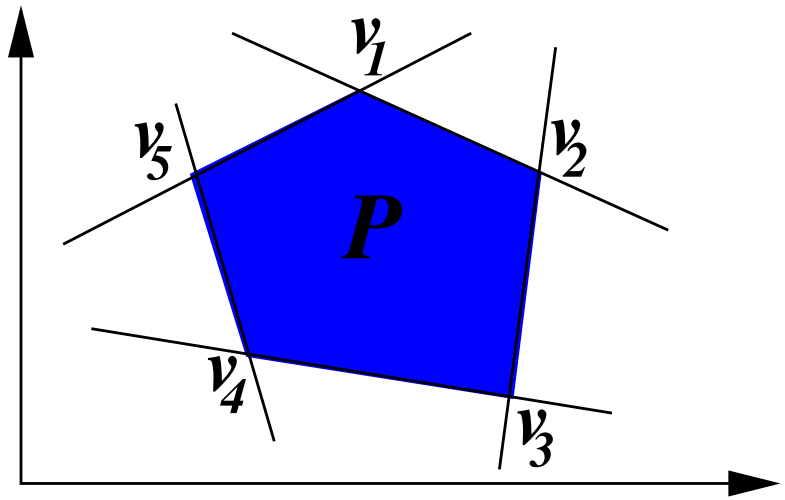
Def.: *O fecho convexo de S, denotado por conv(S) é o conjunto dos pontos que são combinação convexa dos pontos de S.*

Exemplo: Dado dois pontos $x, y \in \mathbb{R}^n$, qualquer ponto que esteja no segmento de reta que liga x a y está em $conv(\{x, y\})$.

Exemplo:



Def.: Os vértices ou pontos extremais de P são os pontos de P que não podem ser escritos como combinação convexa de outros pontos de P



Vértices de P: v_1, v_2, v_3, v_4, v_5

Note que um vértice do poliedro é um ponto único que satisfaz um determinado conjunto de restrições na igualdade.

Determinantes e Resolução de Sistemas Lineares

Determinante por Laplace

Dada matriz A de ordem n e coluna j,

$$|A| = \sum_{i=1}^{n} a_{ij} \cdot (-1)^{i+j} |A^{i,j}|,$$

onde $A^{i,j}$ é a matriz A removendo a linha i e coluna j

Resolução de Sistemas Lineares por Cramer

Dado matriz quadrada A e vetor b, encontrar x tal que $Ax = b$:

$$x_i = \frac{|A_b^j|}{|A|},$$

onde A_b^j é a matriz A com a coluna j trocada pelo vetor b.

Def.: Submatriz de A é uma matriz obtida de A removendo linhas e colunas.

Teorema: *Todo vértice y de um poliedro, definido por uma matriz A, é determinado por uma submatriz não singular A' de A tal que $A' \cdot y = b'$, onde b' contém os correspondentes elementos de b das linhas de A'.*

Def.: Se todo vértice de um poliedro tem apenas componentes inteiras, então o poliedro é dito ser inteiro.

Corolário: *Se P é um poliedro definido por matriz A e vetor b (e.g., $Ax \leq b$), tal que toda submatriz de A tem determinante em $\{-1, 0, +1\}$ e b é inteiro, então todo vértice do poliedro é inteiro.*

Def.: *Uma matriz A é dita ser totalmente unimodular (TU) se para qualquer submatriz quadrada A' de A (selecionando linhas e colunas de A) temos $det(A') \in \{-1, 0, +1\}$.*

Teorema: *Se A é TU, então para todo vetor inteiro b temos que $P = \{x : Ax \leq b\}$ é inteiro.*

Lema: *Se A e B são matrizes TU, então as seguintes matrizes também são TU:*

1. *Matriz obtida de A removendo uma linha (coluna)*
2. *Matriz obtida de A duplicando uma linha (coluna)*
3. *Matriz obtida de A trocando duas linhas (colunas)*
4. A^T *e* $-A$
5. $(A \mid I)$ *e* $\begin{pmatrix} A \\ I \end{pmatrix}$
6. $(A \mid -A)$

Prova. Itens 1., 2., e 3., seguem da propriedade de determinante e de matriz TU.
4. Determinante da transposta é igual ao da original e multiplicar linha por -1 apenas troca sinal do determinante.
5. Basta aplicar regra de Laplace nos elementos da submatriz de I.
6. Duplicar colunas e trocar sinal da coluna mantém TU. □

Teorema: *Se A é TU e b e u são vetores inteiros, então os seguintes poliedros são inteiros:*

1. $\{x : Ax \geq b\}$
2. $\{x : Ax \leq b,\ x \geq 0\}$
3. $\{x : Ax = b,\ x \geq 0\}$
4. $\{x : Ax = b,\ 0 \leq x \leq u\}$

Prova. Note que o poliedro $\{x : Ax = b\}$ é dado por $\{x : Ax \leq b,\ Ax \geq b\}$.

No item 4., basta ver que a matriz $\begin{pmatrix} A \\ -A \\ I \\ -I \end{pmatrix}$ é TU. □

Lema: *Se A é a matriz de incidência de grafo orientado, então A é TU.*

Prova. Por indução na ordem da submatriz. A base é direta.
Seja B uma submatriz quadrada de A.

1. Se há uma coluna z só com elementos nulos: Neste caso o determinante de B é 0.
2. Se há coluna z com exatamente um elemento não nulo $t \in \{-1, +1\}$: Neste caso, o determinante é $+t$ ou $-t$ vezes a submatriz removendo a linha e coluna de t (regra de Laplace).
3. Caso contrário, toda coluna tem -1 e $+1$. Como a soma de todas as linhas é nula, o determinante é nulo.

□

De maneira geral, vale que:

Lema: *Se A é matriz formada por elementos em $\{-1, 0, 1\}$, e cada coluna tem no máximo uma ocorrência de -1 e no máximo uma ocorrência de 1, então A é TU.*

Prova. Exercício. □

Lema: *Se A é a matriz de incidência de um grafo bipartido não-orientado $G = (X, Y, E)$, então A é TU.*

Prova. Multiplique por -1 todas as linhas de A correspondentes aos vértices de Y. Note que isto só muda o sinal dos subdeterminantes.

Isto nos dá uma matriz de incidência de um grafo orientado e a prova segue pelo lema anterior. □

Exercícios:

- Mostre que se A é a matriz de incidência de um grafo orientado e B uma matriz obtida a partir de A transformando alguns elementos não nulos em 0, então B também é TU.
- Seja $G = (N, A)$ um grafo orientado com conjunto de nós N e arcos A e dois nós distintos s e t. Apresente um poliedro definido por variável $x \in [0, 1]^{|A|}$ indexada em A, com número polinomial de restrições, e conjunto de vértices $\mathcal{V}$, tal que, se $\hat{x} \in \mathcal{V}$, então $\hat{x}$ representa os caminhos de s a t, com $\hat{x}_a = 0$ para todo $a \in \delta^-(s) \cup \delta^+(t)$. Isto é, são caminhos sem arcos entrando em s ou saindo de t.

Fluxo de Custo Mínimo

Considere um sistema de produção de bens onde há centros consumidores e produtores.

- Todos os bens produzidos devem ser todos consumidos.
- Os bens produzidos chegam até os consumidores através de rotas.
- Cada rota tem uma capacidade máxima de escoamento (direcionada
- Cada rota tem seu custo para transportar cada unidade do bem.
- Objetivo: Transportar os bens dos produtores para os consumidores minimizando custo para transportar os bens.

Aplicações importantes na Computação:

- Transferência de dados em rede de computadores, deteção de "gargalos" da rede na transferência.
- Projeto de vias de tráfego no planejamento urbano.
- Deteção da capacidade de transmissão entre pontos de redes de telecomunicações.

Notação:

Seja $G = (V, E)$ um grafo não orientado.

- Dado $S \subseteq V$, denotamos por $\delta(S)$ o conjunto de arestas com um extremo em S e outro em $V \setminus S$.
- Dado $v \in V$, denotamos por $\delta(v)$ o conjunto $\delta(\{v\})$.

Seja $D = (V, E)$ um grafo orientado.

- Dado $S \subseteq V$, denotamos por $\delta^+(S)$ o conjunto de arcos com início em S e fim (ponta) em $V \setminus S$.
- Dado $v \in V$, denotamos por $\delta^+(v)$ o conjunto $\delta^+(\{v\})$.
- Dado $S \subseteq V$, denotamos por $\delta^-(S)$ o conjunto de arcos com fim (ponta) em S e início em $V \setminus S$.
- Dado $v \in V$, denotamos por $\delta^-(v)$ o conjunto $\delta^-(\{v\})$.

Definições:

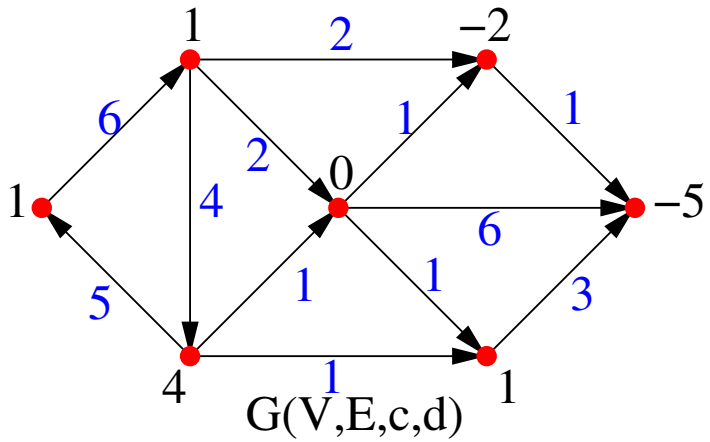
Seja $G = (V, E)$ um grafo orientado com função de capacidades nas arestas $c : E \to \mathbb{Q}^+$, demandas $b : V \to \mathbb{Q}$ e custos nas arestas $w : E \to \mathbb{Q}^+$.

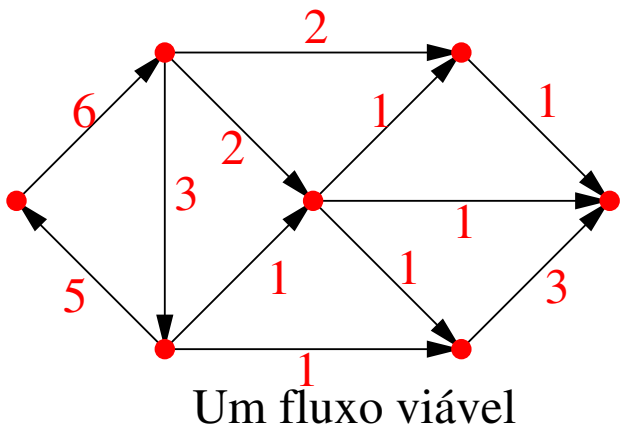
Def.: *Dado um vértice $i \in V$, se $b_i < 0$ dizemos que i é um consumidor e se $b_i > 0$ dizemos que i é um produtor.*

Def.: Dizemos que $x : E \to \mathbb{Q}^+$ é um fluxo em G se

$$\sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = b_v, \quad \forall v \in V, \qquad e \qquad 0 \leq x_e \leq c_e, \quad \forall e \in E$$

Ex.:





Def.: Dado um fluxo $x : E \to \mathbb{Q}^+$ (respeitando c e b) definimos o custo do fluxo x como sendo $\sum_{e \in E} w_e x_e$.

Problema FLUXO DE CUSTO MÍNIMO: Dados um grafo orientado $G = (V, E)$, capacidades $c : E \to \mathbb{Q}^+$, demandas $b : V \to \mathbb{Q}^+$ e uma função de custo nas arestas $w : E \to \mathbb{Q}^+$, encontrar um fluxo $x : E \to \mathbb{Q}^+$ de custo mínimo.

Encontrar x tal que

$$\min \sum_{e \in E} w_e x_e$$

$$\sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = b_v \quad \forall v \in V$$

$$0 \leq x_e \leq c_e \quad \forall e \in E$$

Teorema: *Se as capacidades nas arestas e as demandas dos vértices são inteiros (i.e., $c : E \to \mathbb{Z}^+$ e $b : V \to \mathbb{Z}^+$) então os vértices do poliedro do fluxo são inteiros.*

Prova. Segue do fato que a matriz de incidência de um grafo orientado é TU. □

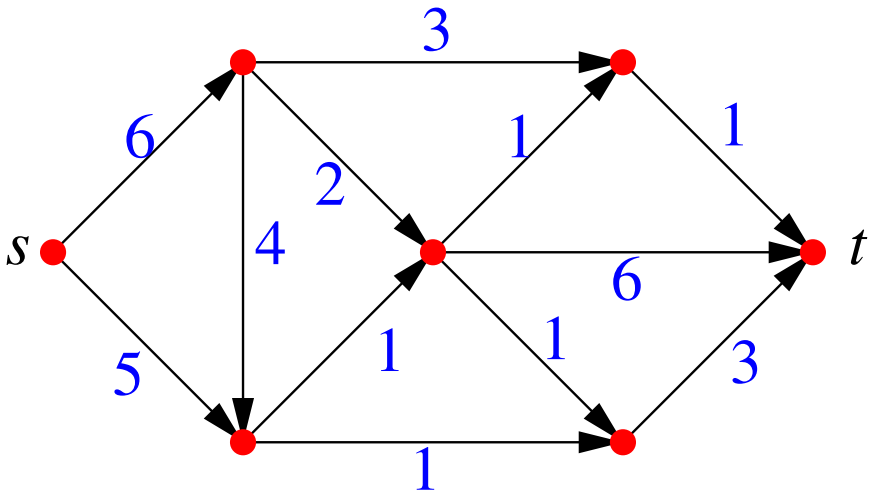
Subproblemas

- Fluxo máximo de um vértice s a um vértice t (st-fluxo)
- Problema do corte de capacidade mínima
- Problema do caminho mínimo (com pesos não negativos)
- Emparelhamento de peso máximo em grafos bipartidos

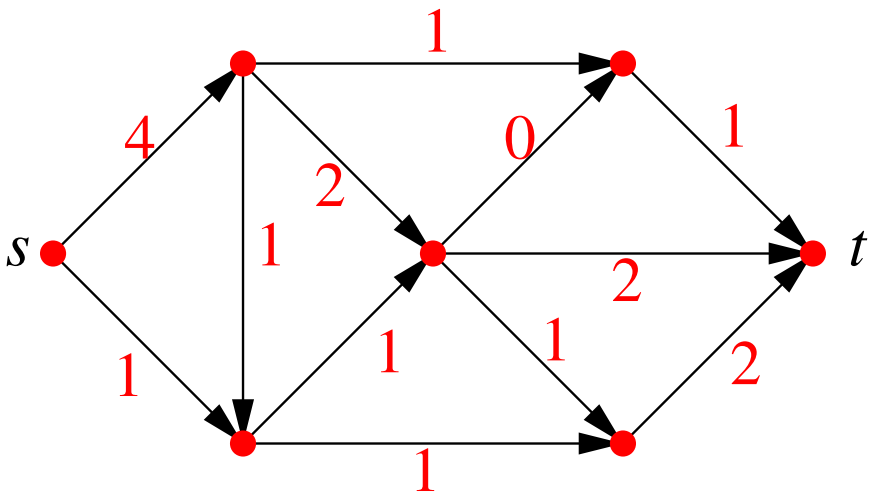
Fluxo Máximo de s para t (st-fluxo máximo)

- Não temos custo para transportar os bens.
- Cada rota tem uma capacidade máxima de escoamento (direcionada).
- Só temos um produtor (vértice s) de produção arbitrariamente grande.
- Só temos um consumidor (vértice t) de consumo arbitrariamente grande.
- Nós internos apenas repassam bens.
- Objetivo: Maximizar o transporte de bens de s para t, respeitando restrições de capacidade do fluxo.

Exemplo:



G(V,E) e capacidades



fluxo de valor 5

Seja $G = (V, E)$ um grafo orientado, capacidades $c : E \to \mathbb{Q}^+$ e vértices s e t.

Def.: *Dizemos que $x : E \to \mathbb{Q}^+$ é um st-fluxo se*

$$\sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = 0 \quad \forall v \in V \setminus \{s, t\}$$

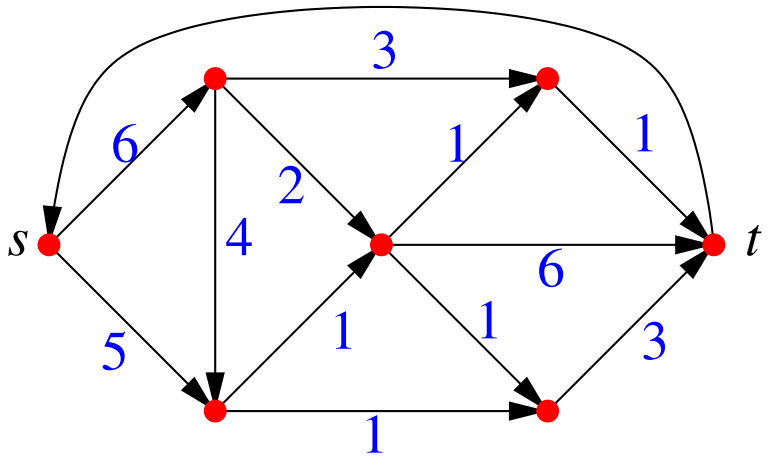
$$0 \leq x_e \leq c_e \quad \forall e \in E$$

Def.: *Dado um st-fluxo x para $G = (V, E, c, s, t)$, definimos o valor do fluxo x como sendo $x(\delta^+(s)) - x(\delta^-(s))$.*

Problema *st*-FLUXO MÁXIMO Dado um grafo orientado $G = (V, E)$, capacidades $c : E \to \mathbb{Q}^+$ e vértices s e t, encontrar um fluxo de s para t de valor máximo.

Formulação Linear

- Para colocar no formato do problema de fluxo de custo mínimo, adicione uma aresta $t \to s$.



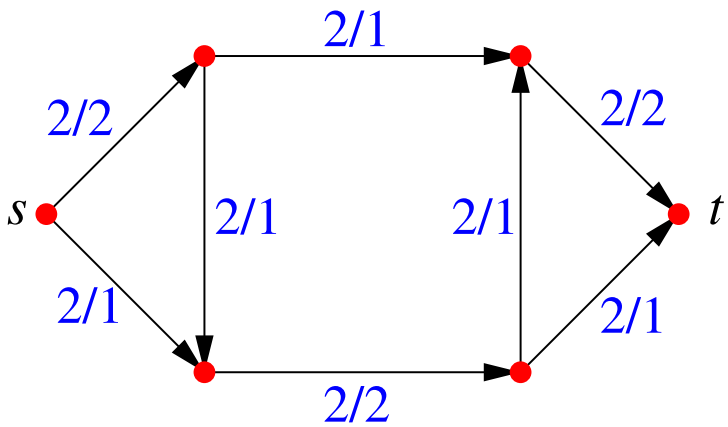
G(V,E) e capacidades

$$\begin{array}{ll} \text{maximize} & x_{ts} \\ \text{sujeito a} & \left\{ \begin{array}{rcl} \sum_{e\in\delta^+(v)} x_e - \sum_{e\in\delta^-(v)} x_e & = & 0 \quad \forall v \in V \\ 0 \quad \leq \quad x_e & \leq & c_e \quad \forall e \in E. \end{array} \right. \end{array}$$

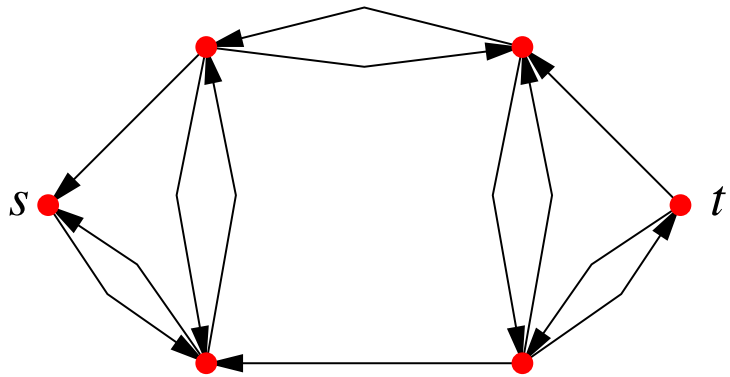
Corolário: *Se as capacidades c_e são inteiras, então os vértices do poliedro do fluxo são inteiros.*

Fluxo Máximo por Caminhos Aumentadores

Grafo Residual Dado um grafo orientado $G = (V, E)$, capacidades $c : E \to \mathbb{Q}^+$, vértices s e t e fluxo f de s para t, o grafo residual é o grafo $G_f = (V, A)$ onde $(u, v) \in A$ sse $(u, v) \in E$ e $f(u, v) < c(u, v)$ ou $(v, u) \in E$ e $f(u, v) > 0$.



Cap/Fluxo (de valor 3)



Grafo residual

Fluxo Máximo por Caminhos Aumentadores

Def.: *Dado um fluxo f, um caminho aumentador de s a t é uma sequência de vértices e arestas $P = (v_1, e_1, v_2, e_2, \ldots, v_{k-1}, e_{k-1}, v_k)$ onde*

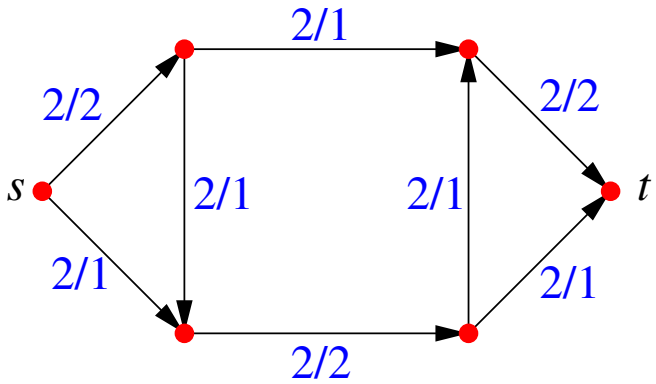
- $s = v_1$ *e* $t = v_k$ *(o caminho vai de s a t)*
- $e_i = (v_i, v_{i+1})$ *ou* $e_i = (v_{i+1}, v_i)$
- *Se* $e_i = (v_i, v_{i+1})$ *então* $f(e) < c(e)$
  (é possível mandar mais fluxo pela aresta e, aumentando o fluxo que chega em v_{i+1}*)*
- *Se* $e_i = (v_{i+1}, v_i)$ *então* $f(e) > 0$
  (é possível diminuir o fluxo pela aresta e, aumentando o fluxo que chega em v_{i+1}*)*

Caminho aumentador dado por um caminho no grafo residual

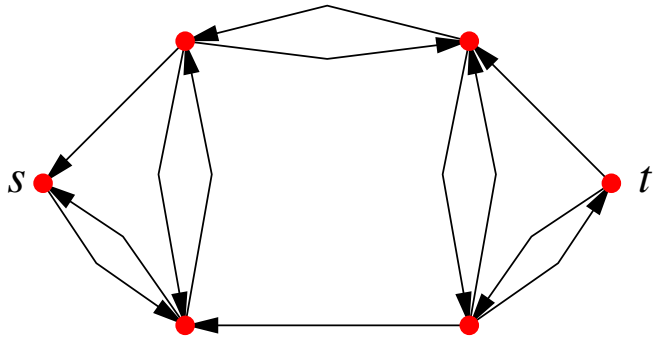
Fluxo Máximo por Caminhos Aumentadores

Idéia: Se existir caminho aumentador de s para t, podemos aumentar o fluxo atual.

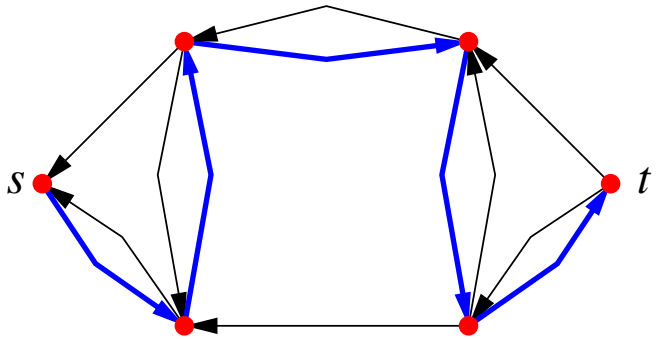
Exemplo: *[de Caminho aumentador]*



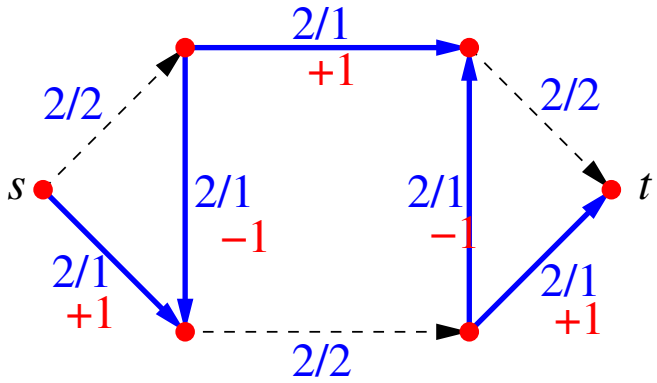
Cap/Fluxo (de valor 3)



Grafo residual



Caminho aumentador

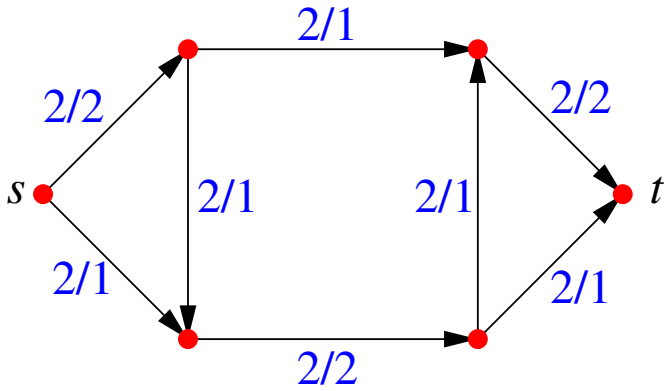


Caminho aumentador de valor 1

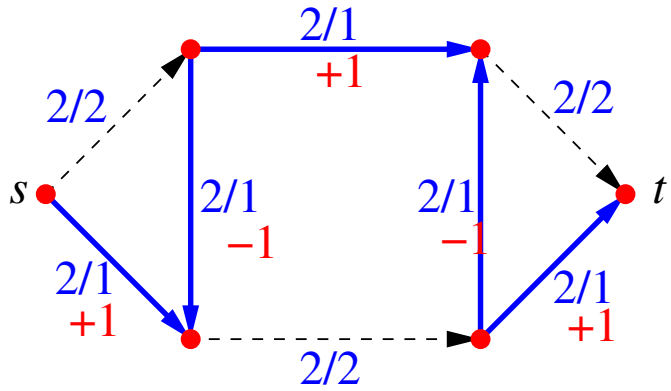
Fluxo Máximo por Caminhos Aumentadores

Idéia: Se existir caminho aumentador de s para t, podemos aumentar o fluxo atual.

Exemplo: *[de Caminho aumentador]*



Cap/Fluxo (de valor 3)



Caminho aumentador de valor 1

Dado caminho aumentador $(s = v_1, e_1, v_2, e_2, \ldots, e_{k-1}, v_k = t)$, defina $\Delta(e_i)$ como

$$\Delta(e_i) = \begin{cases} c(e_i) - f(e_i) & \text{se } e_i = (v_i, v_{i+1}) \\ f(e_i) & \text{se } e_i = (v_{i+1}, v_i) \end{cases}$$

O valor do caminho aumentador é o valor $\min\{\Delta(e_i) : 1 \leq i < k\}$.

Fluxo Máximo por Caminhos Aumentadores

Dado fluxo f, como econtrar um caminho aumentador ?

- Use um algoritmo de busca em largura ou profundidade no grafo residual
- Considerando que cada vértice enxerga os arcos que saem e os que entram nele
- Partindo de s, se a busca atingir t obtenha o valor do caminho aumentador

FORD-FULKERSON

1 Faça $f(e) \leftarrow 0$ para todo $e \in E$.

2 Enquanto existir caminho aumentador P faça

3 Seja $P = (v_1, e_1, v_2, e_2, \ldots, v_{k-1}, e_{k-1}, v_k)$

4 Seja $\Delta(P)$ o valor do caminho aumentador P

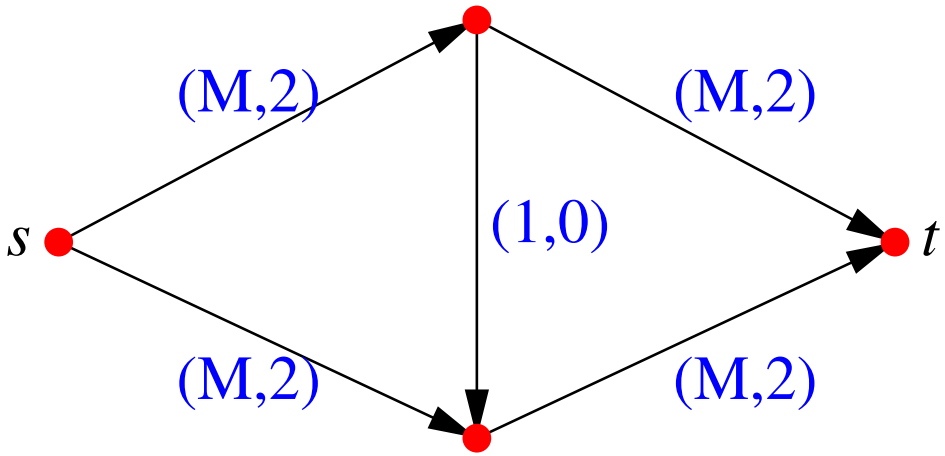
5 Faça $f(e_i) \leftarrow \begin{cases} f(e_i) + \Delta(P) & \text{se } e_i = (v_i, v_{i+1}) \\ f(e_i) - \Delta(P) & \text{se } e_i = (v_{i+1}, v_i) \end{cases}$
para $i = 1, \ldots, k-1$

6 devolva f

Resolvemos o Problema do Fluxo Máximo, usando várias resoluções do Problema da Busca em Grafos (Largura ou Profundidade).

Proposição: *O algoritmo de Ford-Fulkerson não é de tempo polinomial.*

Considere o grafo seguinte, com capacidade M nas arestas, exceto na aresta do meio, que tem capacidade 1:



(c,y)=(capacidade,fluxo viável)

Valor do fluxo atual = 4

A cada iteração, use o caminho aumentador que usa a aresta de capacidade 1. Valor do fluxo máximo: $2M$.

Número de iterações: $O(M)$.

Estratégia Edmonds-Karp

Algoritmo de Edmonds-Karp

- Refinamento do algoritmo Ford-Fulkerson
- Caminhos aumentadores obtidos por busca em largura no grafo residual

Teorema: *O algoritmo de Edmonds-Karp, obtém um fluxo de valor máximo em tempo* $O(VE^2)$.

st-Corte Mínimo

Seja $G = (V, E)$ grafo orientado, capacidades $c : E \to \mathbb{Q}^+$ e vértices s, t

Def.: Um *st*-corte é um conjunto $S \subseteq V$ tal que $s \in S$ e $t \in \overline{S}$ (onde $\overline{S} := V \setminus S$). Também é denotado pelo conjunto de arestas $\delta(S)$.

Def.: Definimos a capacidade de um *st*-corte S, como sendo a soma

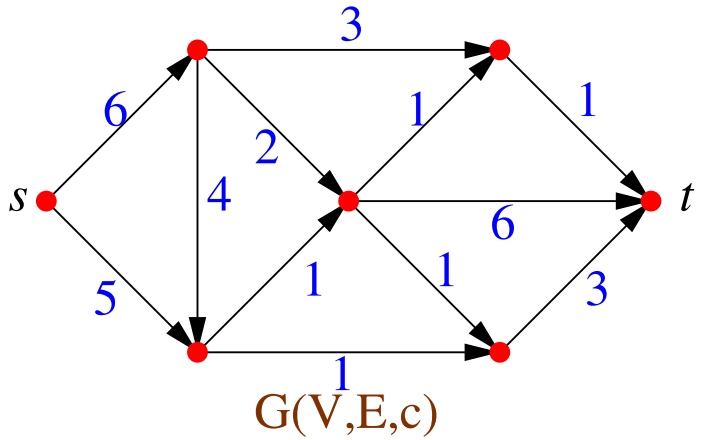
$$c(\delta(S)) := \sum_{e \in \delta(S))} c_e.$$

Problema *st*-CORTE MÍNIMO:
Dado grafo orientado $G = (V, E)$, capacidades $c : E \to \mathbb{Q}^+$ e vértices $s, t \in V$, encontrar um *st*-corte de capacidade mínima.

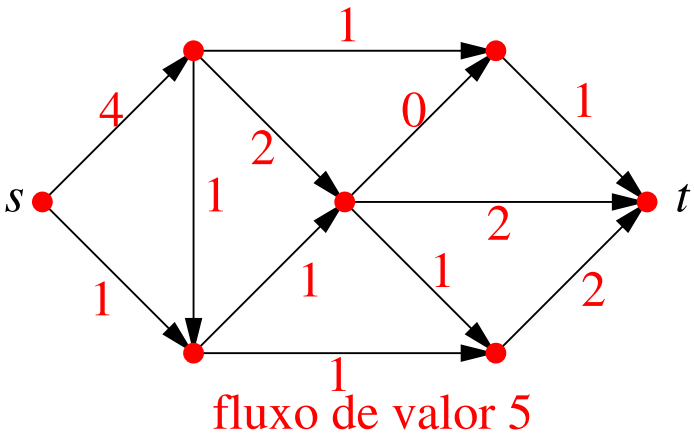
Aplicações importantes na Computação: Projeto de redes de conectividade, Detecção de congestionamentos em redes de conectividade, Classificação de dados (data mining), Clusterização, particionamento de circuitos VLSI, etc

Claramente, um *st*-corte limita superiormente o valor de um fluxo máximo. De fato, vale que:

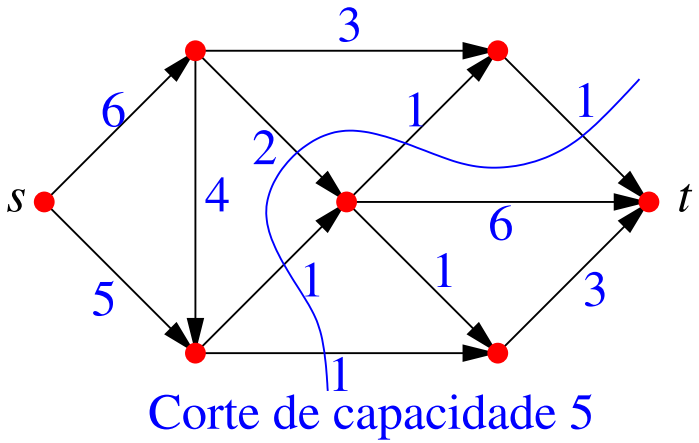
Teorema: *O valor de um st-fluxo máximo de s para t é igual à capacidade de um st-corte mínimo.*



G(V,E,c)



fluxo de valor 5



Corte de capacidade 5

Redução do Corte Mínimo para Fluxo Máximo

Def.: *Um st-corte é um conjunto $S \subset V$ tal que $s \in S$ e $t \notin S$.*

Def.: *Definimos a capacidade de um st-corte S, como sendo a soma*

$$c(\delta^+(S)) := \sum_{e \in \delta^+(S)} c_e.$$

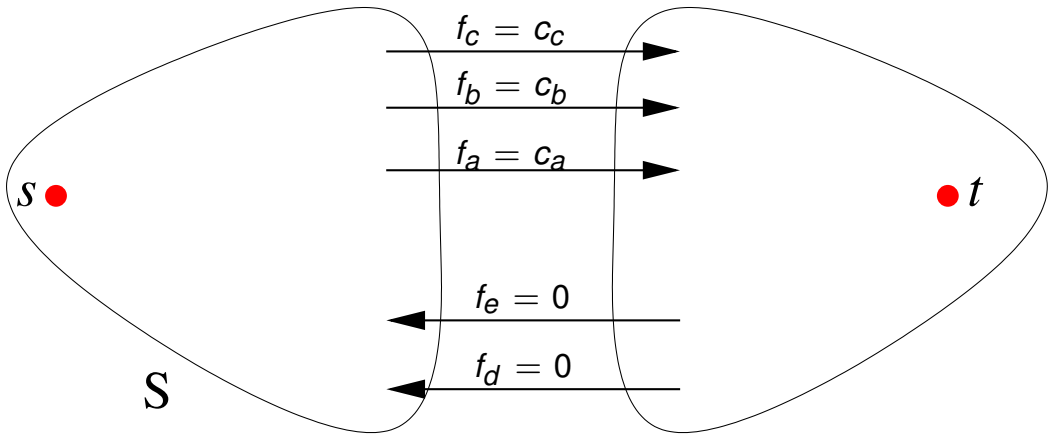
Problema: *[do st-Corte Mínimo] Dado um grafo orientado $G = (V, E)$, capacidades $c : E \to \mathbb{Q}^+$ e vértices s e t, encontrar um st-corte de capacidade mínima.*

Redução do Corte Mínimo para Fluxo Máximo

Teorema: *O algoritmo de Ford-Fulkerson, obtém um fluxo de valor máximo.*

Prova. Exercício. Dicas:

- Considere a última busca realizada para tentar obter um caminho aumentador.
- Esta busca partiu de s, mas não atingiu t.
- Seja S os vértices atingidos pela busca.
- Mostre que S é um corte com capacidade igual ao valor do fluxo encontrado, com a seguinte cara:



Fluxo Máximo $\times$ Corte Mínimo

Do teorema anterior, conclua o seguinte teorema:

Teorema: *[do Fluxo Máximo-Corte Mínimo] O valor do fluxo máximo de s para t é igual à capacidade do menor corte separando s e t.*

Prova. Exercício. Dica: Use a prova que o Algoritmo Ford-Fulkerson obtém fluxo com valor igual ao de um corte. □

Exercício: *Reduza o problema do st-Corte Mínimo para o problema do Fluxo de Valor Máximo.*

Caminhos Disjuntos e Teoremas de Menger

Exercício: Apresente algoritmos de tempo polinomial:

Problema: *[Caminhos Disjuntos em Grafos orientados] Dado grafo orientado e vértices s e t, encontrar o maior número de st-caminhos orientados disjuntos nos arcos.*

Problema: *[Corte Separador de s e t em grafos orientados] Dado grafo orientado e vértices s e t, encontrar o menor número de arcos cuja remoção desconecta todos os st-caminhos.*

Problema: *[Caminhos Disjuntos em Grafos não-orientados] Dado grafo não-orientado e vértices s e t, encontrar o maior número de st-caminhos disjuntos nas arestas.*

Problema: *[Corte Separador de s e t em grafos não-orientados] Dado grafo não-orientado e vértices s e t, encontrar o menor número de arestas cuja remoção desconecta todos os st-caminhos.*

Caminhos Disjuntos e Teoremas de Menger

Exercícios: Apresente algoritmos de tempo polinomial:

Problema: *[Caminhos Disjuntos nos Vértices em Grafos orientados] Dado grafo orientado e vértices s e t, encontrar o maior número de st-caminhos orientados disjuntos nos vértices.*

Problema: *[Corte de Vértices que separa s e t em grafos orientados] Dado grafo orientado e vértices s e t, encontrar o menor número de vértces cuja remoção desconecta todos os st-caminhos.*

Problema: *[Caminhos Disjuntos nos Vértices em Grafos não-orientados] Dado grafo não-orientado e vértices s e t, encontrar o maior número de st-caminhos disjuntos nos vértices.*

Problema: *[Corte de Vértices que separa s e t em grafos não-orientados] Dado grafo não-orientado e vértices s e t, encontrar o menor número de vértices cuja remoção desconecta todos os st-caminhos.*

Exercícios:

- Faça um programa que, dado um *st*-fluxo de valor máximo, encontra um *st*-corte mínimo em tempo linear.
- Dado um grafo *não orientado*, com capacidades nas arestas, indicando a capacidade de fluxo que pode ir de um extremo ao outro de uma aresta, em qualquer uma das direções. O problema do fluxo máximo de um vértice s a um vértice t é encontrar um fluxo que sai de s e chega em t, respeitando a capacidade nas arestas e a lei de conservação do fluxo em cada vértice interno. Reduza este problema ao problema de encontrar um *st*-fluxo máximo em grafo orientado.
- Dado um grafo não orientado com capacidades nas arestas, e dois vértices s e t, resolva o problema de encontrar um corte de capacidade mínima separando s e t.

Caminhos Disjuntos e Teoremas de Menger

Exercício: Apresente algoritmos de tempo polinomial

Def.: *Um grafo conexo não orientado G é k-aresta conexo (resp. k-vértice conexo) se a remoção de menos que k arestas (resp. vértices) quaisquer nos dá um grafo conexo.*

Problema: *Mostre que um grafo não orientado é k-aresta conexo (resp. k-vértice conexo) se e somente se há k caminhos aresta disjuntos (resp. vértice disjuntos) entre quaisquer par de vértices.*

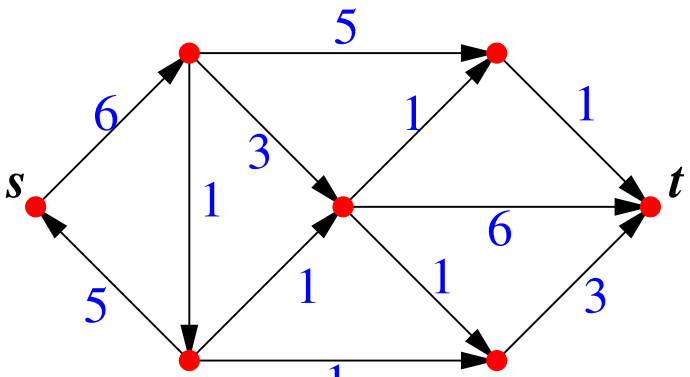
Problema: *Decida se um grafo não orientado é k-aresta conexo (resp. k-vértice conexo), caso contrário, apresente conjunto com menos que k arestas (resp. vértices) cuja remoção nos dá um grafo desconexo.*

Problema: *Formule a versão orientada do problema acima e proponha um algoritmo que o resolva em tempo polinomial.*

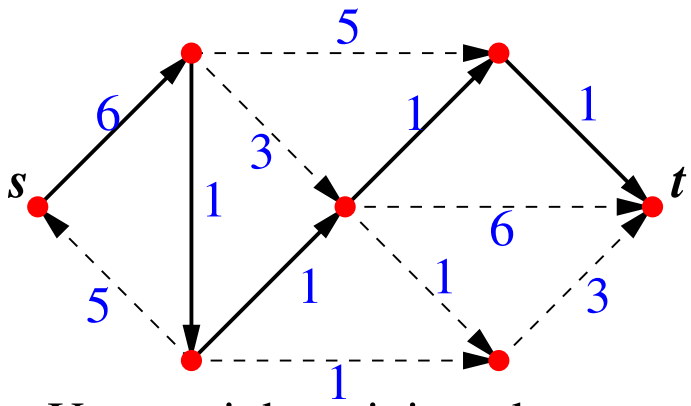
Um excelente livro sobre problemas em fluxo é dado em Ahuja, Magnanti e Orlin; Network Flows: Theory, Algorithms and Applications

Problema do Caminho Mínimo

Problema CAMINHO MÍNIMO: Dado um grafo orientado $G = (V, E)$, custos nas arestas $w : E \to \mathbb{Q}^+$ e vértices s e t, encontrar um caminho de s para t de custo mínimo.



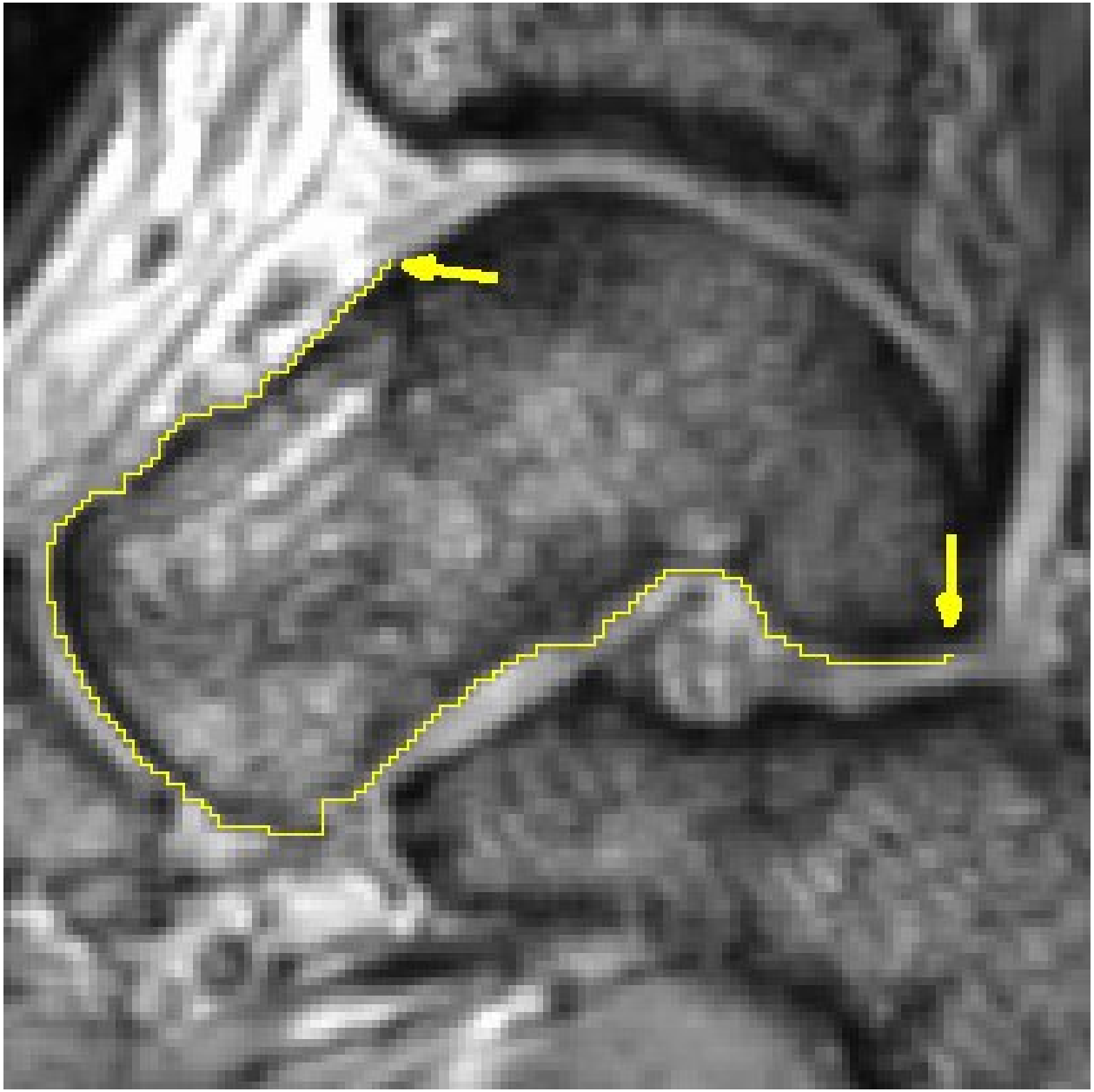
G(V,E,w)



Um caminho minimo de s para t

Aplicações: Determinação de rotas de custo mínimo, segmentação de imagens, Escolha de centro distribuidor, reconhecimento de fala, etc

Exemplo de aplicação em segmentação de imagens



Formulação para o Problema do Caminho Mínimo

Considere a seguinte formulação:

$$(P) \quad \begin{array}{ll} \text{minimize} & \sum_{e \in E} w_e x_e \\ \text{sujeito a} & \begin{cases} \sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = 0 & \forall v \in V \setminus \{s,t\} \\ \sum_{e \in \delta^+(s)} x_e - \sum_{e \in \delta^-(s)} x_e = 1 & \\ \sum_{e \in \delta^+(t)} x_e - \sum_{e \in \delta^-(t)} x_e = -1 & \\ 0 \leq x_e \leq 1 & \forall e \in E. \end{cases} \end{array}$$

- Formulação é caso particular do problema do fluxo de custo mínimo

Corolário: *Se x é um ponto extremal ótimo de (P), então x é inteiro. I.e., $x_e \in \{0,1\} \forall e \in E$.*

Teorema: *Se x é um ponto extremal ótimo de (P), então as arestas $e \in E$ onde $x_e = 1$ formam um caminho mínimo ligando s a t.*

Caminhos Disjuntos de Custo Mínimo

Exercícios:

1. Considere a seguinte formulação para o problema do caminho mínimo:

$$(P) \quad \begin{array}{ll} \min & \sum_{e \in E} w_e x_e \\ \text{s.a.} & \left\{ \begin{array}{rcl} \sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e & = & 0 \quad \forall v \in V \setminus \{s,t\} \\ \sum_{e \in \delta^+(s)} x_e & = & 1 \\ \sum_{e \in \delta^-(t)} x_e & = & 1 \\ 0 \leq x_e \leq 1 & & \forall e \in E \setminus (\delta^-(s) \cup \delta^+(t)) \\ 0 \leq x_e \leq 0 & & \forall e \in \delta^-(s) \cup \delta^+(t) \end{array} \right. \end{array}$$

   Mostre que o poliedro associado a esta formulação é inteiro.

2. Se x é um ponto extremal ótimo de (P), então as arestas $e \in E$ onde $x_e = 1$ formam um caminho mínimo ligando s a t.

Caminhos Disjuntos de Custo Mínimo

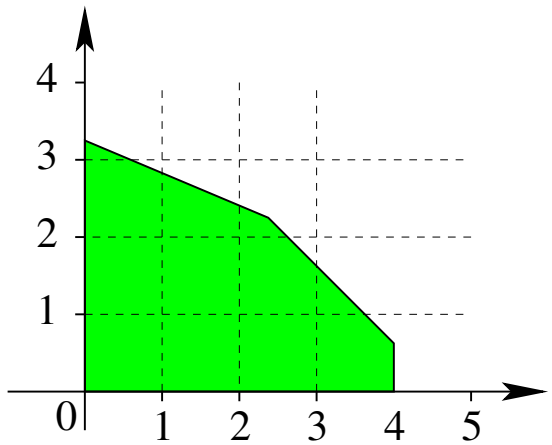
Exercícios:

1. Considere um grafo não-orientado $G = (V, E, w)$, com pesos nas arestas $w : E \to \mathbb{Q}^+$, vértices s e t e inteiro positivo k. Projete um algoritmo de tempo polinomial que encontra k caminhos disjuntos nas arestas, $P_1, \ldots, P_k$, que ligam o vértice s ao vértice t, e cujo peso total das arestas é mínimo; ou, se for o caso, escreva que não existem estes k caminhos.
2. Descreva o problema do exercício anterior, mas para grafos orientados. Projete um algoritmo de tempo polinomial para resolvê-lo.
3. Descreva os problemas dos dois exercícios anteriores, mas para caminhos disjuntos nos vértices internos (os caminhos só se interseptam nos vértices s e t).

Def.: *Dado um poliedro P, definimos seu fecho inteiro, P_I, como sendo o fecho convexo dos vetores inteiros de P. I.e.,*

$$P_I := conv\{x \in P : \ x \text{ é inteiro}\}$$

Exemplo: *Exemplo de poliedro P, conjunto I dos pontos inteiros em P e o fecho convexo de I.*



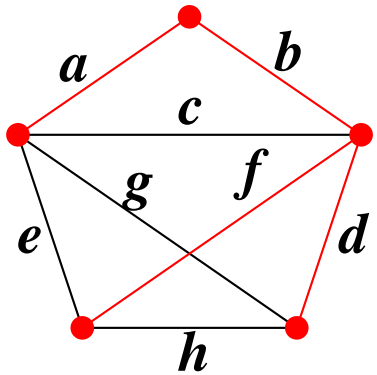
Poliedro P Pontos inteiros I $P_I := conv(I)$

Def.: *Dado um conjunto finito e ordenado E, dizemos que $\chi^A := (\chi^A_e : e \in E)$ é o vetor de incidência de A, $A \subseteq E$; onde $\chi^A_e = 1$ se $e \in A$ e 0 caso contrário.*

Exemplo: Se $E = (a, b, c, d, e, f, g)$ e $A = \{a, c, e, f\}$ e χ^A é um vetor de incidência de A em E

$$\chi^A = (1, 0, 1, 0, 1, 1, 0).$$

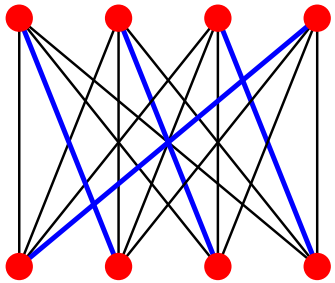
Exemplo: Seja $G = (V, E)$ o grafo abaixo com ordenação das arestas $E = (a, b, c, d, e, f, g, h)$ e T o subgrafo definido pelas arestas em vermelho.



Então o vetor de incidência das arestas de T em E é $\chi^T = (1, 1, 0, 1, 0, 1, 0)$.

Emparelhamento em Grafos Bipartidos

Def.: Dado um grafo $G = (V, E)$, dizemos que $M \subseteq E$ é um emparelhamento de G se M não tem arestas com extremos em comum.



Problema EMPARELHAMENTO-BIPARTIDO: Dados um grafo bipartido $G = (V, E)$, e custos nas arestas $c : E \to \mathbb{Z}$, encontrar emparelhamento $M \subseteq E$ que maximize $c(M)$.

Aplicações: Encontrar atribuição de candidatos para vagas maximizando aptidão total, atribuição de motoristas de veículos, formação de equipes (eg. com um chefe e subordinados), etc.

Exercício: *Reduza o problema de encontrar um Emparelhamento de Cardinalidade Máxima em grafos bipartidos ao Problema do Fluxo Máximo. Mostre que a aplicação do Algoritmo Ford-Fulkerson nos dá um algoritmo de tempo polinomial para este problema.*

Formulação para Emparelhamento em Grafos Bipartidos

Formulação em Programação Linear Inteira

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} c_e x_e \\ \text{sujeito a} \quad & \begin{cases} \sum_{e \in \delta(v)} x_e \leq 1 & \forall v \in V \\ 0 \leq x_e \leq 1 & \forall e \in E \\ x_e \text{ inteiro} & \forall e \in E \end{cases} \end{aligned}$$

Note que a formulação acima só não é um programa linear, devido a restrição de integralidade em x_e.

Relaxação Linear

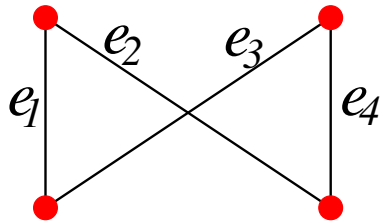
$$\begin{array}{lll} & \text{maximize} & \sum_{e \in E} c_e x_e \\ (LP_{Emp}) & \text{sujeito a} & \begin{cases} \sum_{e \in \delta(v)} x_e & \leq & 1 \quad \forall v \in V \\ 0 \leq x_e \leq 1 & & \forall e \in E \end{cases} \end{array}$$

Teorema: *Os vértices do poliedro LP_{Emp} são inteiros.*

Prova. Exercício. □

Exemplo:

Considere o seguinte grafo bipartido com custos unitários:



Programa Linear

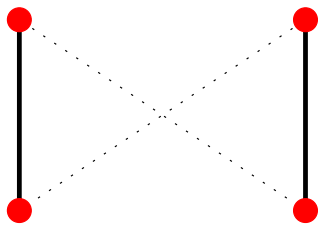
$$\begin{aligned} \text{maximize} \quad & x_{e_1} + x_{e_2} + x_{e_3} + x_{e_4} \\ \text{sujeito a} \quad & \begin{cases} x_{e_1} + x_{e_2} \leq 1, \\ x_{e_3} + x_{e_4} \leq 1, \\ x_{e_1} + x_{e_3} \leq 1, \\ x_{e_2} + x_{e_4} \leq 1, \\ 0 \leq x_{e_1} \leq 1, \\ 0 \leq x_{e_2} \leq 1, \\ 0 \leq x_{e_3} \leq 1, \\ 0 \leq x_{e_4} \leq 1 \end{cases} \end{aligned}$$

Variáveis do programa linear correspondente:

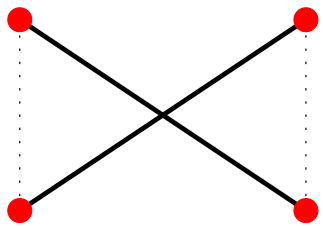
$$X = (x_{e_1}, x_{e_2}, x_{e_3}, x_{e_4})$$

Os seguintes vetores são soluções ótimas do programa linear:

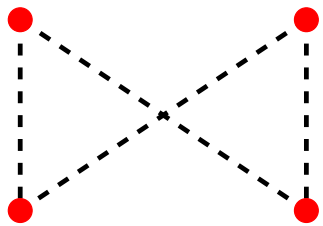
$$X' = (1,0,0,1) \quad X'' = (0,1,1,0) \quad X''' = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$$



X'



X''



X'''

- X' e X'' são vértices do poliedro do emparelhamento
- X''' é solução ótima, mas é combinação convexa de X' e X'' ($X''' = \frac{1}{2}X' + \frac{1}{2}X''$). I.e., X''' não é vértice de P_{Emp}.

Problema do Transporte

Problema PROBLEMA DO TRANSPORTE: São dados um conjunto A de fornecedores e um conjunto B de consumidores, onde $A = \{1, \ldots, m\}$ e $B = \{1, \ldots, n\}$, todos relativos a um mesmo material. O fornecedor $i \in A$ produz a_i unidades e o consumidor $j \in B$ consome b_j unidades. O custo para transportar uma unidade do fornecedor i para o consumidor j é de c_{ij}, para todo $i \in A$ e $j \in B$. Considere que $T = \sum_i a_i = \sum_j b_j$. O problema consiste em determinar a forma mais barata para transportar T unidades dos fornecedores para os consumidores, satisfazendo a produção e consumo dos produtores e consumidores.

Aplicações: Transporte de materiais, abastecimento (energia, água,...), transmissão de dados/broadcast

Problema do Transporte

Exercício: Resolva o Problema do Transporte.

Exercício: Resolva uma variação do Problema do Transporte (capacitado nas arestas) supondo que o fornecimento x_{ij} é no máximo um valor u_{ij}, dado na entrada, para todo $i \in A$ e $j \in B$.

Exercício: Resolva uma variação do Problema do Transporte supondo que cada aresta tem uma capacidade máxima, $\sum_{i \in A} a_i \geq \sum_{j \in B} b_j$ (não necessariamente valendo na igualdade) devemos atender todos os consumidores, mas os fornecedores só precisam respeitar capacidade de produção (não necessariamente, a produção de um fornecedor i precisa ser consumida).

Exercício: b-matching: Dados grafo bipartido $G = (A, B, E)$ com função de peso nas arestas $w : E \to \mathbb{Q}$, e função limitadora de grau $b : A \cup B \to Z^+$, encontrar um conjunto de arestas M tal que o número de arestas de M incidentes a um vértice i é no máximo b_i e o peso total das arestas de M é máximo.

Exercícios

Exercício: Dados grafo bipartido $G = (V, E)$ com função de peso nos vértices $w : V \to \mathbb{Q}$, encontrar um conjunto de vértices $I \subseteq V$ que não tem arestas incidentes em comum (conjunto independente). Apresente uma formulação em programação linear inteira para resolver este problema, cuja relaxação apresente apenas vértices inteiros. Com isso, projete um algoritmo de tempo polinomial para encontrar um conjunto independente em G de peso máximo.

Exercício: Dados grafo bipartido $G = (V, E)$ com função de peso nos vértices $w : V \to \mathbb{Q}$, encontrar um conjunto de vértices $C \subseteq V$ tal que toda aresta $e \in E$ tem pelo menos um dos extremos em C (cobertura por vértices). Faça como no exercício anterior e projete um algoritmo de tempo polinomial para encontrar uma cobertura por vértices em G de peso mínimo. Outra maneira de resolver o problema da cobertura por vértices, é reduzindo este problema para o anterior. Mostre como isto pode ser feito.

Exercícios

Exercício: Dados grafo bipartido $G = (V, E)$ com função de peso nos vértices $w : V \to \mathbb{Q}$, encontrar um conjunto de arestas $F \subseteq E$ tal que para todo vértice $v \in V$, há pelo menos uma aresta de F incidente em v (cobertura por arestas). Faça como no exercício anterior e projete um algoritmo de tempo polinomial para encontrar uma cobertura por vértices em G de peso mínimo.

Dualidade em Programação Linear

Considere o seguinte PL:

$$
\begin{array}{ll}
\text{minimize} & c_1x_1 + c_2x_2 + c_3x_3 \\
\text{sujeito a} & \left\{ \begin{array}{lcl}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 & \geq & b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 & = & b_2 \\
a_{31}x_1 + a_{32}x_2 + a_{33}x_3 & \leq & b_3 \\
x_1 \geq 0, \qquad\qquad x_3 \leq 0 &&
\end{array} \right.
\end{array}
$$

Vamos delimitar o valor ótimo do LP:
Multiplique as restrições por $y_1 \geq 0$, y_2 e $y_3 \leq 0$:

$$
\begin{array}{lcl}
y_1(a_{11}x_1 + a_{12}x_2 + a_{13}x_3) & \geq & y_1b_1 \\
y_2(a_{21}x_1 + a_{22}x_2 + a_{23}x_3) & = & y_2b_2 \\
y_3(a_{31}x_1 + a_{32}x_2 + a_{33}x_3) & \geq & y_3b_3
\end{array}
$$

Somando estas inequações obtemos:

$$
\begin{array}{lcl}
y_1(a_{11}x_1 + a_{12}x_2 + a_{13}x_3) & \geq & y_1 b_1 \\
y_2(a_{21}x_1 + a_{22}x_2 + a_{23}x_3) & = & y_2 b_2 \\
y_3(a_{31}x_1 + a_{32}x_2 + a_{33}x_3) & \geq & y_3 b_3 \\
\hline
(y_1 a_{11} + y_2 a_{21} + y_3 a_{31})x_1 + & & \\
(y_1 a_{12} + y_2 a_{22} + y_3 a_{32})x_2 + & & \\
(y_1 a_{13} + y_2 a_{23} + y_3 a_{33})x_3 & \geq & (y_1 b_1 + y_2 b_2 + y_3 b_3) = yb
\end{array}
$$

Comparando com a função objetivo $cx = c_1 x_1 + c_2 x_2 + c_3 x_3$, se

$$c_1 x_1 \geq (y_1 a_{11} + y_2 a_{21} + y_3 a_{31})x_1$$

$$c_2 x_2 = (y_1 a_{12} + y_2 a_{22} + y_3 a_{32})x_2$$

$$c_3 x_3 \geq (y_1 a_{13} + y_2 a_{23} + y_3 a_{33})x_3$$

Nestas condições, temos $cx \geq yb$ e portanto yb é limitante de cx

Como $x_1 \geq 0$ e $x_3 \leq 0$, podemos simplificar as condições para ter $cx \geq yb$ como:

$$c_1 \geq y_1 a_{11} + y_2 a_{21} + y_3 a_{31}$$

$$c_2 = y_1 a_{12} + y_2 a_{22} + y_3 a_{32}$$

$$c_3 \leq y_1 a_{13} + y_2 a_{23} + y_3 a_{33}$$

Naturalmente, queremos dentre todos os valores de y, o que melhor delimita $cx \geq yb$, i.e., com yb máximo. Assim, obtemos o seguinte problema dual:

$$\begin{array}{ll} \text{maximize} & y_1 b_1 + y_2 b_2 + y_3 b_3 \\ \text{sujeito a} & \begin{cases} y_1 a_{11} + y_2 a_{21} + y_3 a_{31} \leq c_1 \\ y_1 a_{12} + y_2 a_{22} + y_3 a_{32} = c_2 \\ y_1 a_{13} + y_2 a_{23} + y_3 a_{33} \geq c_3 \\ y_1 \geq 0, \qquad\qquad y_3 \leq 0 \end{cases} \end{array}$$

Convenção para nomear estes sistemas:

Problema Primal

$$\begin{array}{ll} \text{minimize} & c_1x_1 + c_2x_2 + c_3x_3 \\ \text{sujeito a} & \begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 \leq b_3 \\ x_1 \geq 0, \qquad x_3 \leq 0 \end{cases} \end{array}$$

Problema Dual

$$\begin{array}{ll} \text{maximize} & y_1b_1 + y_2b_2 + y_3b_3 \\ \text{sujeito a} & \begin{cases} y_1a_{11} + y_2a_{21} + y_3a_{31} \leq c_1 \\ y_1a_{12} + y_2a_{22} + y_3a_{32} = c_2 \\ y_1a_{13} + y_2a_{23} + y_3a_{33} \geq c_3 \\ y_1 \geq 0, \qquad y_3 \leq 0 \end{cases} \end{array}$$

Exercício: *Faça exatamente a mesma análise feita anteriormente para se obter um dual (como limitante de um programa linear), mas em vez de partir de um problema de minimização, comece com um problema de maximização, e obtenha seu dual como um problema de minimização.*

Exercício: *Faça o programa dual da formulacao relaxada dos seguintes problemas:*

- *Problema da Cobertura de Vertices*
- *Problema da Cobertura por Conjuntos*
- *Problema de Localizacao de Facilidades*
- *Problema de Steiner*

Problemas Primal e Dual

Problema Primal

$$
(P)\qquad
\begin{array}{lrcll}
\text{minimize} & cx & & & \\
\text{sujeito a} & (Ax)_i & \geq & b_i & \text{para cada } i \text{ em } M_1\,, \\
 & (Ax)_i & = & b_i & \text{para cada } i \text{ em } M_2\,, \\
 & (Ax)_i & \leq & b_i & \text{para cada } i \text{ em } M_3\,, \\
 & x_j & \geq & 0 & \text{para cada } j \text{ em } N_1\,, \\
 & x_j & \leq & 0 & \text{para cada } j \text{ em } N_3\,.
\end{array}
$$

Problema Dual

$$
(D)\qquad
\begin{array}{lrcll}
\text{maximize} & yb & & & \\
\text{sujeito a} & (yA)_j & \leq & c_j & \text{para cada } j \text{ em } N_1\,, \\
 & (yA)_j & = & c_j & \text{para cada } j \text{ em } N_2\,, \\
 & (yA)_j & \geq & c_j & \text{para cada } j \text{ em } N_3\,, \\
 & y_i & \geq & 0 & \text{para cada } i \text{ em } M_1\,, \\
 & y_i & \leq & 0 & \text{para cada } i \text{ em } M_3\,.
\end{array}
$$

Lema: *Seja (P) um programa primal de minimização e (D) seu problema dual de maximização, então $cx \geq yb$ para todo $x \in P$ e $y \in D$.*

Folgas Complementares

Dois vetores x e y, indexados por M e N respectivamente, têm folgas complementares se,

$x_j = 0 \quad$ ou $\quad (yA)_j = c_j \quad \forall j \in N_1 \cup N_3$ (folgas complementares primais)
e
$y_i = 0 \quad$ ou $\quad (Ax)_i = b_i \quad \forall i \in M_1 \cup M_3$ (folgas complementares duais).

Lema: *(das folgas complementares) Se (P) é um programa linear e (D) seu programa dual, x e y soluções viáveis de (P) e (D) então*

$$(cx = yb) \quad \Leftrightarrow \quad (x \text{ e } y \text{ satisfazem folgas complementares})$$

Teorema: *(da dualidade) Se* (P) *é um programa linear de minimização e* (D) *seu programa dual (de maximização), então vale exatamente uma das possibilidades:*

1. (P) *e* (D) *são viáveis e* OPT-LP(P)=OPT-LP(D).
2. (P) *é viável e* (D) *é inviável e* OPT-LP(P) $= -\infty$.
3. (P) *é inviável e* (D) *é viável e* OPT-LP(D) $= \infty$.
4. (P) *e* (D) *são inviáveis.*

Lema: *(de Farkas) Exatamente um dos programas restritos têm solução:*

$$(RP) \quad \begin{array}{rcll} \exists & x & \in & \mathbb{Q}^N \\ (Ax)_i & \geq & b_i & \forall i \in M, \\ x_i & \geq & 0 & \forall j \in N. \end{array} \qquad (RD) \quad \begin{array}{rcll} \exists & y & \in & \mathbb{Q}^M \\ yb & > & 0 & \\ (yA)_j & \leq & 0 & \forall j \in N, \\ y_i & \geq & 0 & \forall i \in M. \end{array}$$

Prova.

Considere o programa linear (P) e seu dual (D):

$$(P) \quad \begin{array}{rcll} \min & 0x & & \\ (Ax)_i & \geq & b_i & \forall i \in M, \\ x_i & \geq & 0 & \forall j \in N. \end{array} \qquad (D) \quad \begin{array}{rcll} \max & yb & & \\ (yA)_j & \leq & 0 & \forall j \in N, \\ y_i & \geq & 0 & \forall i \in M. \end{array}$$

Note que (P) é viável se e só se (RP) é víavel e o programa (D) é sempre viável pois $y = 0$ satisfaz as restrições.

Como (D) é viável, apenas as alternativas 1. ou 3. do Lema da Dualidade podem ocorrer.

Caso 1: (P) é viável e OPT-LP(P)=OPT-LP(D).

(P) viável $\Rightarrow$ (RP) é viável.

$\Big(\forall y \in (D),\ yb \leq \text{OPT-LP}(D) = \text{OPT-LP}(P) = 0\Big) \Rightarrow (RD)$ é inviável.

Caso 3: OPT-LP$(D) = \infty$ e (P) é inviável.

(P) inviável $\Rightarrow$ (RP) é inviável.

$\Big(\text{OPT-LP}(D) = \infty\Big) \Rightarrow \Big(\exists y \in (D) :\ yb > 0\Big) \Rightarrow (RD)$ é viável. □

Método Primal-Dual

- Baseado em Folgas Complementares e Problemas de Viabilidade
- Muitas vezes produz algoritmos combinatórios

Método Primal-Dual Clássico

$$(P) \quad \begin{array}{rcll} \min & cx & & \\ (Ax)_i & \geq & b_i & \forall i \in M, \\ x_i & \geq & 0 & \forall j \in N. \end{array} \qquad (D) \quad \begin{array}{rcll} \max & yb & & \\ (yA)_j & \leq & c_j & \forall j \in N, \\ y_i & \geq & 0 & \forall i \in M. \end{array}$$

Folgas Complementares

Se $\hat{x}$ e $\hat{y}$ são soluções ótimas de (P) e (D), então

$\hat{x}_j = 0 \quad$ ou $\quad (\hat{y}A)_j = c_j \qquad$ (folgas complementares primais)

$\hat{y}_i = 0 \quad$ ou $\quad (A\hat{x})_i = b_i \qquad$ (folgas complementares duais)

Estratégia:

Dado vetor y viável de (D),

- obter x viável de (P) satisfazendo folgas complementares com y
  ou
- obter y' tq. $y'' \leftarrow y + y'$ é viável em (D) e $y''b > yb$;
  repita processo com y''

Folgas Complementares

Se $\hat{x}$ e $\hat{y}$ são soluções ótimas de (P) e (D), então

$$\hat{x}_j = 0 \quad \text{ou} \quad (\hat{y}A)_j = c_j \qquad \text{(folgas complementares primais)}$$
$$\hat{y}_i = 0 \quad \text{ou} \quad (A\hat{x})_i = b_i \qquad \text{(folgas complementares duais)}$$

Dado vetor y viável de (D),

$$I(y) := \{i \in M : \ y_i = 0\} \quad \text{e} \quad J(y) := \{j \in N : \ (yA)_j = c_j\}\,.$$

$$\text{(Viabil.)} \quad \begin{array}{rcll} (Ax)_i & \geq & b_i & \forall i \in M\,, \\ x_i & \geq & 0 & \forall j \in N\,. \end{array} \quad + \quad \text{(F.C.)} \quad \begin{array}{rcll} (Ax)_i & = & b_i & \forall i \in M \setminus I(y)\,, \\ x_j & = & 0 & \forall j \in N \setminus J(y)\,. \end{array}$$

Problema Restrito Primal

$$(RP) \quad \begin{array}{rcll} (Ax)_i & \geq & b_i & \forall i \in I(y)\,, \\ (Ax)_i & = & b_i & \forall i \in M \setminus I(y)\,, \\ x_j & \geq & 0 & \forall j \in J(y)\,, \\ x_j & = & 0 & \forall j \in N \setminus J(y)\,. \end{array}$$

Pelo Lema de Farkas, (RP) é viável ou (RD) é viável, não ambos (exercício).

$$
(RP)\quad \begin{array}{rcll} (Ax)_i & \geq & b_i & \forall i \in I(y)\,, \\ (Ax)_i & = & b_i & \forall i \in M \setminus I(y)\,, \\ x_j & \geq & 0 & \forall j \in J(y)\,, \\ x_j & = & 0 & \forall j \in N \setminus J(y)\,. \end{array} \quad \oplus \quad (RD)\quad \begin{array}{rcll} y'b & > & 0 & \\ (y'A)_j & \leq & 0 & \forall j \in J(y)\,, \\ y'_i & \geq & 0 & \forall i \in I(y)\,. \end{array}
$$

$$I(y) := \{i \in M : y_i = 0\} \quad \text{e} \quad J(y) := \{j \in N : (yA)_j = c_j\}.$$

$$(D) \quad \begin{array}{rl} \max & yb \\ (yA)_j & \leq c_j \quad \forall j \in N, \\ y_i & \geq 0 \quad \forall i \in M. \end{array}$$

$$(RD) \quad \begin{array}{rcll} y'b & > & 0 & \\ (y'A)_j & \leq & 0 & \forall j \in J(y), \\ y'_i & \geq & 0 & \forall i \in I(y). \end{array}$$

Lema: *Se y é viável para (D) e y' é viável para (RD), então, existe $\theta > 0$ tal que $y'' \leftarrow y + \theta y'$ é também é viável para (D).*

Prova. Exercício. □

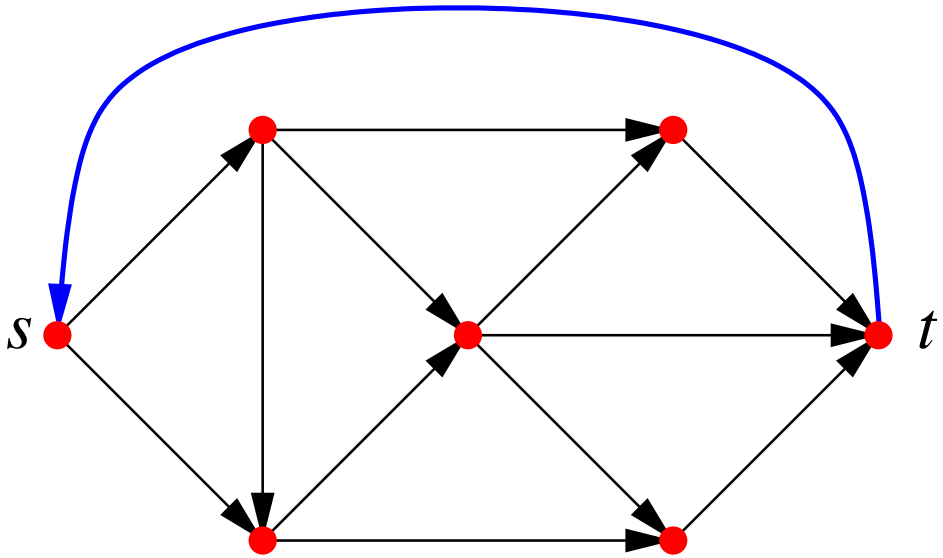
Método PRIMAL-DUAL (A, b, c)

```
1   seja y um vetor viável de (D)
2   enquanto RP(A, b, y) não tem solução faça
3       seja y' uma solução do RD(A, b, y)
4       se y + θy' em (D) para todo θ positivo
5           então devolva y'
6           senão seja θ máximo tal que y + θy' é viável em (D)
7                   y ← y + θy'
8   seja x uma solução do RP(A, b, y)
9   devolva x e y
```

Problema do Fluxo Máximo

Problema FLUXO-MÁXIMO: Dado um grafo orientado $G = (N, A)$, capacidades $c : A \to \mathbb{N}$ e vértices s e t, encontrar um fluxo de valor máximo de s para t.

Simplificação: adicionar aresta (t, s) sem restrição de capacidade.



Objetivo: Encontrar fluxo que respeita conservação de fluxo em todos os vértices e maximiza fluxo na aresta (t, s).

Por conveniência, vamos chamar a formulação do fluxo de Dual.

$$
(D) \quad \begin{array}{lrcll} \max & y_{ts} & & & \\ & \sum_{e\in\delta^+(i)} y_e - \sum_{e\in\delta^-(i)} y_e & = & 0 & \forall i \in N, \\ & y_{ij} & \leq & c_{ij} & \forall ij \in A, \\ & y_{ij} & \geq & 0 & \forall ij \in A. \end{array}
$$

Primal: encontrar $x = x' || x''$, x' indexado por N e x'' indexado por A que

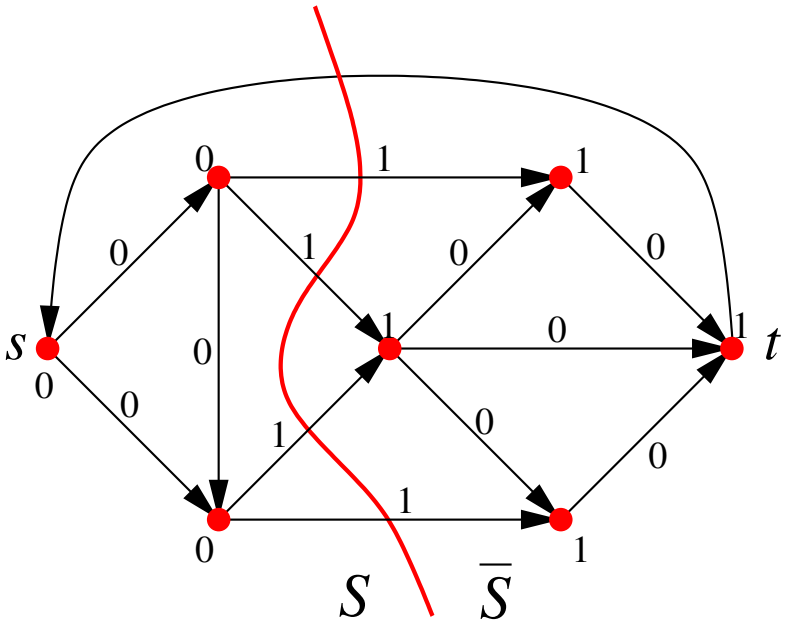
$$
(P) \quad \begin{array}{lrcll} \min & \sum_{ij\in A} c_{ij}\, x''_{ij} & & & \\ & x'_t - x'_s & \geq & 1 & , \\ & x'_i - x'_j + x''_{ij} & \geq & 0 & \forall ij \in A, \\ & x''_{ij} & \geq & 0 & \forall ij \in A. \end{array}
$$

Sobre o programa *(P)*

$$
(P) \quad \begin{aligned} \min \quad & \sum_{ij \in A} c_{ij} x''_{ij} \\ & x'_t - x'_s \geq 1 \quad , \\ & x'_i - x'_j + x''_{ij} \geq 0 \quad \forall ij \in A, \\ & x''_{ij} \geq 0 \quad \forall ij \in A. \end{aligned}
$$

Dado um corte $S \subset N$ que separa s de t (i.e. $s \in S$ e $t \notin S$), defina

$$
x'_i = \begin{cases} 0 & \forall i \in S \\ 1 & \forall i \in N \setminus S \end{cases} \qquad x''_{ij} = \begin{cases} 1 & \forall ij \in \delta^+(S) \\ 0 & \forall ij \in A \setminus \delta^+(S) \end{cases}
$$



Temos $x = x' \| x''$ viável para (P) definindo um corte com capacidade dada pela função objetivo.

Problemas Restritos

Dado y, fluxo viável seja

$$I := \{ij \in A : y_{ij} = 0\} \quad \text{e} \quad J := \{ij \in A : y_{ij} = c_{ij}\}.$$

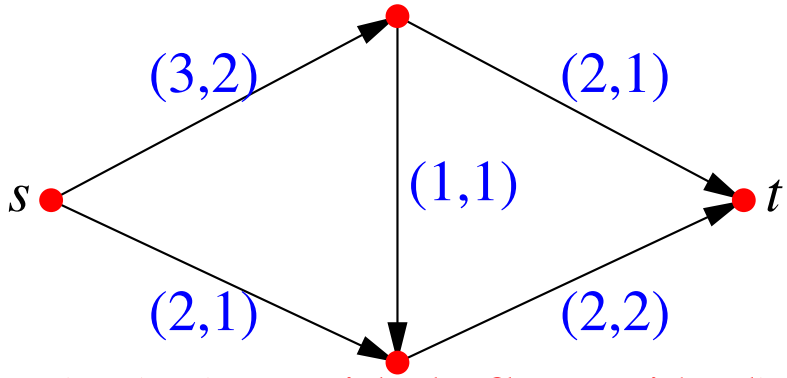
Restrito Primal: encontrar $x = x' || x''$ viável em (D) satisfazendo F.C.

$$(RP) \quad \begin{array}{rcll} x'_t - x'_s & \geq & 1 & , \\ x'_i - x'_j + x''_{ij} & = & 0 & \forall ij \in A \setminus I, \\ x'_i - x'_j + x''_{ij} & \geq & 0 & \forall ij \in I, \\ x''_{ij} & \geq & 0 & \forall ij \in J, \\ x''_{ij} & = & 0 & \forall ij \in A \setminus J. \end{array}$$

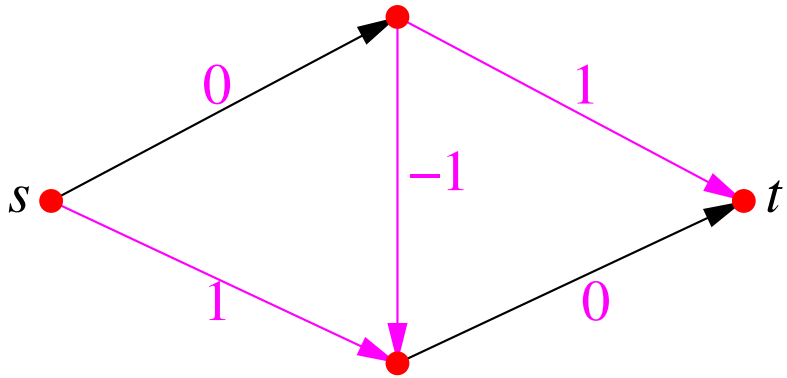
Restrito Dual: Se (RP) é inviável, pelo Lema de Farkas, (RD) é viável

$$(RD) \quad \begin{array}{rcll} y_{ts} & > & 0 & , \\ \displaystyle\sum_{e \in \delta^+(i)} y_e - \sum_{e \in \delta^-(i)} y_e & = & 0 & \forall i \in N, \\ y_{ij} & \leq & 0 & \forall ij \in J, \\ y_{ij} & \geq & 0 & \forall ij \in I. \end{array}$$

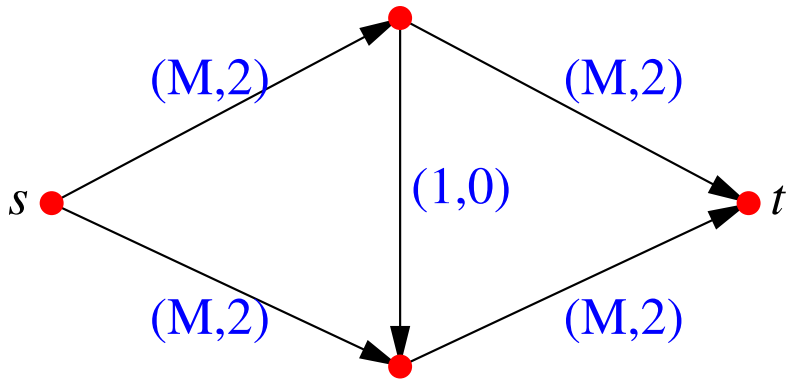
(RD) viável $\Rightarrow \exists$ caminho aumentador $P = (y')$ de s para t, representando um fluxo adicional.



(c,y)=(capacidade,fluxo viável)
Valor do fluxo inicial = 3



Caminho aumentador
Fluxo no caminho = 1



(c,y)=(capacidade,fluxo viável)

Valor do fluxo atual = 4

(RD) inviável $\Rightarrow$ (RP) é viável

Seja $S := \{v \in N :$ existe caminho aumentador de s para $v\}$ e

$$x'_i = \begin{cases} 0 & \forall i \in S \\ 1 & \forall i \in N \setminus S \end{cases} \qquad x''_{ij} = \begin{cases} 1 & \forall ij \in \delta^+(S) \\ 0 & \forall ij \in A \setminus \delta^+(S) \end{cases}$$

Temos que $x = x' \| x''$ é viável em (RP)

Programação Linear Inteira

Os problemas resolvidos anteriormente por programação linear

- puderam ser resolvidos de maneira eficiente (tempo polinomial)

É possível usar programação linear na resolução de problemas NP-difíceis ?

SIM!!!

- Vários problemas tem sido bem resolvidos através de métodos em PLI.

Problema PLI: Dados matriz $A = (a_{ij}) \in \mathbb{Q}^{n \times n}$, vetores $c = (c_i) \in \mathbb{Q}^n$ e $b = (b_i) \in \mathbb{Q}^m$, encontrar vetor $x = (x_i) \in \mathbb{Z}^n$ (se existir) que

$$\begin{array}{ll} \text{minimize} & c_1 x_1 + c_2 x_2 + \cdots + c_m x_m \\ \text{sujeito a} & \left\{ \begin{array}{rcl} a_{11} x_1 + a_{12} x_2 + \cdots + a_{1m} x_n & \leq & b_1 \\ a_{21} x_1 + a_{22} x_2 + \cdots + a_{2m} x_n & \geq & b_2 \\ \vdots & \vdots & \vdots \\ a_{n1} x_1 + a_{n2} x_2 + \cdots + a_{nm} x_n & = & b_n \\ x_i \in \mathbb{Z} & & \end{array} \right. \end{array}$$

Qual a diferença com programação linear ?

Sobre a complexidade computacional, as diferenças são grandes.

Teorema: *PLI é um problema NP-difícil.*

Prova. Exercício: Reduza um dos problemas NP-difíceis vistos anteriormente para a forma de um problema PLI. □

Estratégias para resolver problemas NP-difíceis através de PLI:

- por arredondamento
  - modelar o problema como problema de programação linear inteira
  - relaxar a formulação para programação linear
  - resolver o programa linear
  - arredondar as variáveis para cima/baixo, de acordo com o problema.
- pelo método branch & bound
  - modelar o problema como problema de programação linear inteira
  - relaxar a formulação para programação linear
  - enumerar o espaço de soluções através do método branch & bound. Cada nó da árvore representa um programa linear. Cada programa linear (nó) é ramificado enquanto houver valores fracionários em sua solução.

- pelo método de planos de corte
  - modelar o problema como problema de programação linear inteira
  - relaxar a formulação para programação linear
  - repetidamente inserir uma desigualdade válida que separa o ponto fracionário.
  - parar quando obtiver uma solução inteira
- pelo método branch & cut
  - combinação do método branch & bound e
  - método de planos de corte

O ponto de partida de todas estas estratégias é modelar como um problema de programação linear inteira

Modelagem de Problemas

Modelagem através de variáveis 0/1

- Um dos passos mais importantes para se resolver um problema por programação linear inteira é a escolha da formulação.

Como no problema de emparelhamento, vamos formular alguns problemas NP-difíceis através de variáveis 0/1.
Em geral, a idéia principal é

- definir variáveis $0/1$, digamos x_i, $i = 1, \ldots, n$, tal que
- se $x_i = 1$ então o objeto i pertence à solução
- se $x_i = 0$ então o objeto i não pertence à solução

Formulação do Problema da Mochila

Problema MOCHILA: Dados itens $S = \{1, \ldots, n\}$ com valor v_i e tamanho s_i inteiros, $i = 1, \ldots, n$, e inteiro B, encontrar $S' \subseteq S$ que maximiza $\sum_{i \in S'} v_i$ tal que $\sum_{i \in S'} s_i \leq B$.

Vamos definir soluções através de variáveis binárias x_i, $i \in S$ tal que
$x_i = 1$ indica que o elemento i pertence à solução e
$x_i = 0$ indica que o elemento i não pertence à solução.

$$\begin{array}{ll} \text{maximize} & \displaystyle\sum_{i \in [n]} v_i x_i \\ \text{sujeito a} & \left\{ \begin{array}{rcll} \displaystyle\sum_{i \in [n]} s_i x_i & \leq & B & \\ x_i & \in & \{0,1\} & \forall i \in [n] \end{array} \right. \end{array}$$

onde $[n] = \{1, \ldots, n\}$.

Coberturas, Empacotamentos e Partições

Coberturas, Empacotamentos e Partições são condições que ocorrem frequentemente na formulação de problemas.

Seja E um conjunto e $\mathcal{C}$ uma coleção de subconjuntos de E.

Seja $\mathcal{C}_e := \{C \in \mathcal{C} : e \in C\}$ e $\mathcal{S} \subseteq \mathcal{C}$ tal que $x_C = 1 \Leftrightarrow C \in \mathcal{S}$.

- $\mathcal{S}$ **é uma cobertura se**

$$\sum_{C \in \mathcal{C}_e} x_C \geq 1 \quad \forall e \in E,$$

- $\mathcal{S}$ **é um empacotamento se**

$$\sum_{C \in \mathcal{C}_e} x_C \leq 1 \quad \forall e \in E,$$

- $\mathcal{S}$ **é uma partição se**

$$\sum_{C \in \mathcal{C}_e} x_C = 1 \quad \forall e \in E.$$

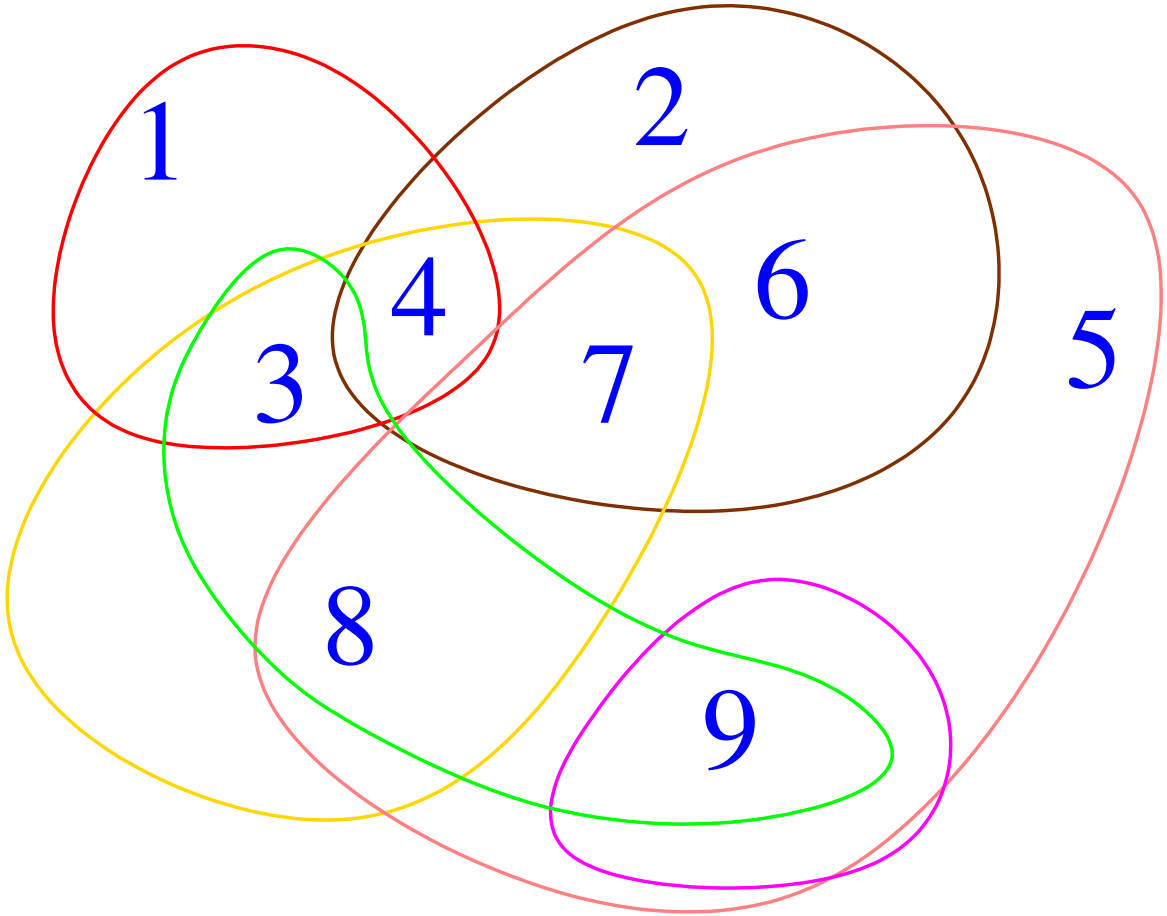
Problema da Cobertura por Conjuntos

Def.: Dada uma coleção $\mathcal{S}$ de subconjuntos de E dizemos que $\mathcal{S}$ *cobre* E, ou é uma *cobertura* de E, se $\cup_{S \in \mathcal{S}} S = E$.

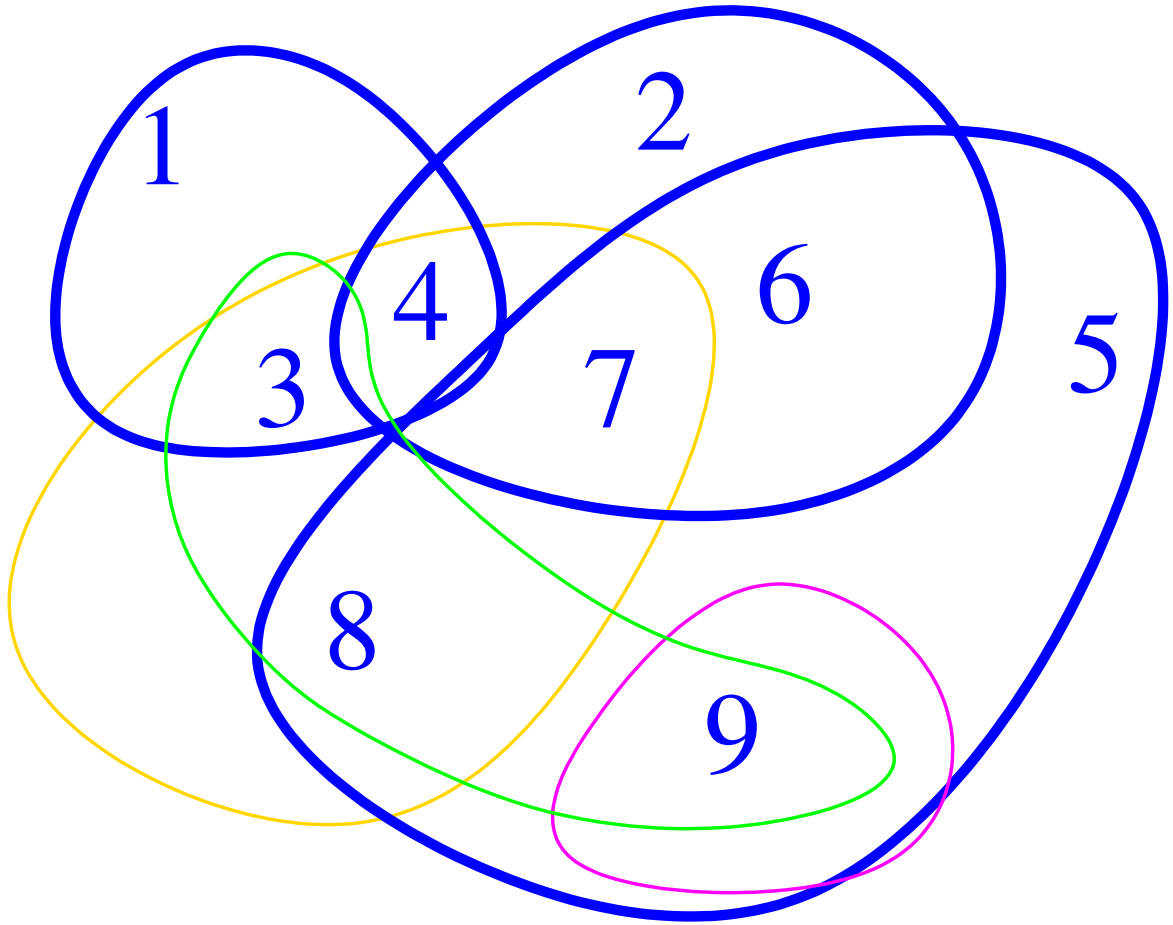
Problema COBERTURA POR CONJUNTOS: Dados conjunto E, subconjuntos $\mathcal{S}$ de E, custos $c(S)$, $S \in \mathcal{S}$, encontrar cobertura $\mathcal{S}' \subseteq \mathcal{S}$ que minimiza $c(\mathcal{S}')$.

Teorema: COBERTURA POR CONJUNTOS é NP-difícil.

Encontre uma cobertura por conjuntos:



Cobertura por conjuntos:



Deteção de Virus

Reportado por Williamson'98 de estudo feito na IBM:

- Para cada vírus determinamos algumas seqüências de bytes (assinaturas), cada seqüência com 20 ou mais bytes.
- Total de 9000 seqüências para todos os virus.
- O objetivo é encontrar o menor conjunto de seqüências que detecta todos os 5000 vírus.

Caso particular do problema de cobertura por conjuntos:

- Conjuntos: Cada seqüência determina um conjunto de vírus que contém a seqüência.
- Conjunto Base: Conjunto de todos os vírus.
- Objetivo: Encontrar uma cobertura por conjuntos de cardinalidade mínima.

Solução encontrada: 180 seqüências para detectar todos os 5000 virus.

Formulação do Problema da Cobertura por Conjuntos

Vamos definir soluções através de variáveis binárias x_S, $S \in \mathcal{S}$ tal que $x_S = 1$ indica que o conjunto S pertence à solução e $x_S = 0$ indica que o conjunto S não pertence à solução.

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{S \in \mathcal{S}} c_S x_S \\ \text{sujeito a} & \begin{cases} \displaystyle\sum_{S \in \mathcal{S}_e} x_S \geq 1 & \forall e \in E \\ x_S \in \{0,1\} & \forall S \in \mathcal{S}, \end{cases} \end{array}$$

onde $\mathcal{S}_e$ é definido como $\mathcal{S}_e := \{S \in \mathcal{S} : e \in S\}$.

A primeira restrição diz que para qualquer elemento do conjunto E, pelo menos um dos conjuntos que o cobrem (conjuntos $\mathcal{S}_e$) deve pertencer a solução.

Problema da Localização de Recursos

Problema LOCALIZAÇÃO DE RECURSOS (FACILITY LOCATION PROBLEM): Dados potenciais recursos $F = \{1, \ldots, n\}$, clientes $C = \{1, \ldots, m\}$, custos f_i para instalar o recurso i e custos $c_{ij} \in \mathbb{Z}$ para um cliente j ser atendido pelo recurso i. Encontrar recursos $A \subseteq F$ minimizando custo para instalar os recursos em A e atender todos os clientes

Aplicação: Instalar postos de distribuição de mercadorias, centros de atendimento, instalação de antenas em telecomunicações, etc.

Note que neste problema temos de determinar quais os recursos que iremos instalar e como conectar os clientes aos recursos instalados.

Temos dois tipos de custos envolvidos:

- Se resolvermos instalar o recurso, devemos pagar pela sua instalação.
- Devemos pagar um preço pela conexão estabelecida entre um cliente e o recurso instalado mais próximo.

Vamos definir soluções através de variáveis binárias y_i $i \in F$ tal que
$y_i = 1$ indica que o recurso i foi escolhido para ser instalado e
$y_i = 0$ indica que o recurso i não vai ser instalado.
e variáveis x_{ij}, tal que
$x_{ij} = 1$ indica que o cliente j será conectado ao recurso i
$x_{ij} = 0$ indica que o cliente j não será conectado ao recurso i.

$$\text{minimize} \quad \sum_{i \in F} f_i y_i + \sum_{ij \in E} c_{ij} x_{ij}$$

$$\text{sujeito a} \quad \begin{cases} \sum_{ij \in E} x_{ij} = 1 & \forall j \in C, \\ x_{ij} \leq y_i & \forall ij \in E, \\ y_i \in \{0,1\} & \forall i \in F \\ x_{ij} \in \{0,1\} & \forall i \in F \text{ e } j \in C. \end{cases}$$

A primeira restrição indica que todo cliente deve ser conectado a algum recurso.
A segunda restrição indica que um cliente só deve ser conectado a um recurso instalado.

Problema da Floresta de Steiner

Seja G um grafo e $\mathcal{R}$ uma coleção de subconjuntos de V_G.

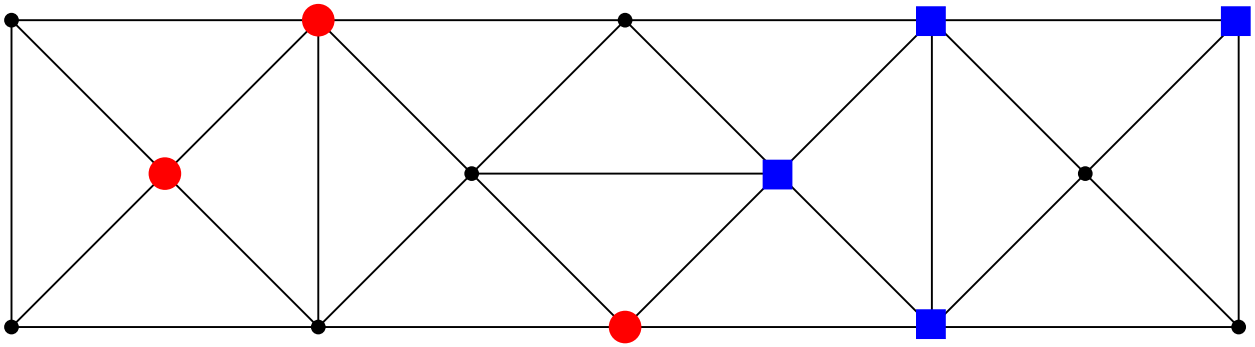
Def.: *Uma $\mathcal{R}$-floresta de G é qualquer floresta geradora F de G tal que todo elemento de $\mathcal{R}$ está contido em algum componente de F.*

Problema Floresta de Steiner: Dados um grafo G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e e uma coleção $\mathcal{R}$ de subconjuntos de V_G, encontrar uma $\mathcal{R}$-floresta F que minimize $c(F)$.

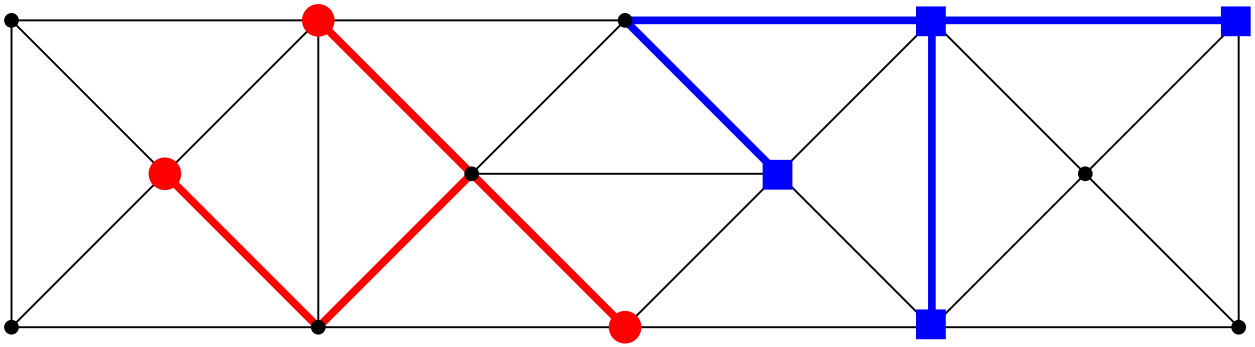
Aplicações:

- Roteamento em circuitos VLSI (VLSI Layout and routing).
- Projeto de redes de conectividade.
- Determinação de amplificadores de sinal em redes óticas.
- Construção de árvores filogenéticas.

Exemplo: Considere o seguinte grafo com possíveis ligações. Por simplicidade, definimos restrições de conectividade para nós representados pelos mesmos símbolos. Assim, devemos encontrar uma solução de custo mínimo que conecte todos os círculos e que conecte todos os quadrados.



Possível solução



Formulação:

Em muitos problemas que envolvem alguma estrutura de conectividade, é comum usarmos variáveis binárias para as arestas de conexão, pois

1. em geral as arestas tem custos que estamos querendo minimizar/maximizar
2. permitem facilmente escrever restrições relativas às condições de conectividade a que devem respeitar.

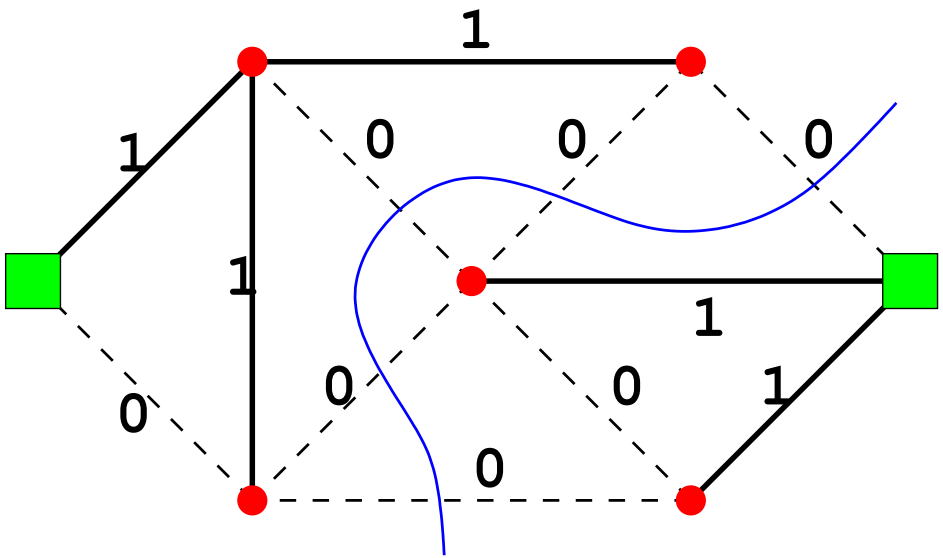
Vamos definir soluções através de variáveis binárias x_{ij} tal que
$x_{ij} = 1$ indica que a aresta ij pertence à solução
$x_{ij} = 0$ indica que a aresta ij não pertence à solução
Note que se S é inválido

- podemos dividir o conjunto de vértices em duas partes, I e J tal que
  - $i \in I$, $j \in J$ e
  - nenhuma aresta de S liga I e J.
  - Isto nos dá um corte que separa i e j.

Se em algum momento temos uma atribuição para x que ainda não é solução, então

- poderemos encontrar um conjunto de arestas que formam este corte separador e
- pelo menos uma aresta do corte separador deve estar na solução

Exemplo de atribuição para x_e (para cada aresta e) que não é solução viável.



Note que pelo menos uma das arestas que pertence ao corte (aresta que corta linha azul) deve pertencer à solução.

Vc se lembra de como encontrar um corte que separa dois vértices s e t de capacidade mínima ?

Formulação:

$$\text{minimize} \quad \sum_{e \in E} c_e x_e$$

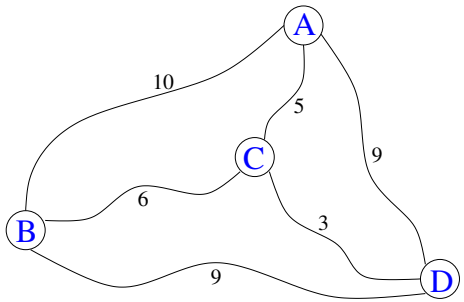
$$\text{sujeito a} \quad \begin{cases} \sum_{e \in \delta(S)} x_e \geq 1 & \forall S \subset V : \\ & S \text{ separa algum conjunto de termi} \\ & \text{(desigualdades de corte)} \\ x_e \in \{0,1\} & \forall e \in E \end{cases}$$

dizemos que S separa um conjunto de terminais se existe $R \in \mathcal{R}$ tal que $S \cap R \neq \emptyset$ e $R \setminus S \neq \emptyset$

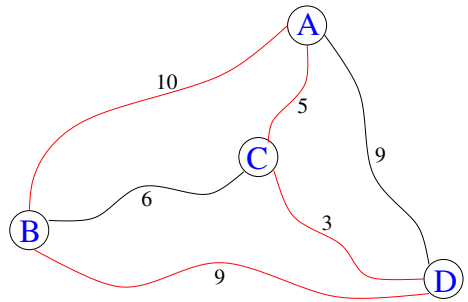
A primeira restrição impõe que todo corte separador deve ter pelo menos uma aresta que pertença à solução.

Problema do Caixeiro Viajante

Problema TSP: Dados um grafo $G = (V, E)$ e um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e, determinar um circuito hamiltoniano C que minimize $c(C)$.



Grafo G



Circuito Hamiltoniano de G de custo 27

Aplicações:

- Perfuração e solda de circuitos impressos.
- Determinação de rotas de custo mínimo.
- Seqüênciamento de DNA (Genoma).
- Outras: `http://www.math.princeton.edu/tsp/apps`
- D.S. Johnson: TSP Chalenging: `www.research.att.com/~dsj/cht`
- CONCORDE: Applegate, Bixby, Chvátal, Cook'01: Branch & cut para TSP

Vamos definir soluções através de variáveis binárias x_{ij} tal que
$x_{ij} = 1$ indica que a aresta ij pertence ao circuito da solução
$x_{ij} = 0$ indica que a aresta ij não pertence ao circuito da solução

$$\text{minimize} \quad \sum_{e \in E} c_e x_e$$

$$\text{sujeito a} \quad \begin{cases} \sum_{e \in \delta(v)} x_e = 2 & \forall v \in V \\ \sum_{e \in \delta(S)} x_e \geq 2 & \forall S \subset V, \quad S \neq \emptyset \\ & \text{(restrições de subcircuito)} \\ x_e \in \{0,1\} & \forall e \in E \end{cases}$$

A primeira restrição diz que para todo vértice há duas arestas da solução que incidem no vértice (uma para entrar e outra para sair).

A segunda restrição diz que a solução deve ser conexa.
Note que se não colocarmos as desigualdades da segunda restrição, a solução gerada poderia ser um conjunto de circuitos.

Exercícios:

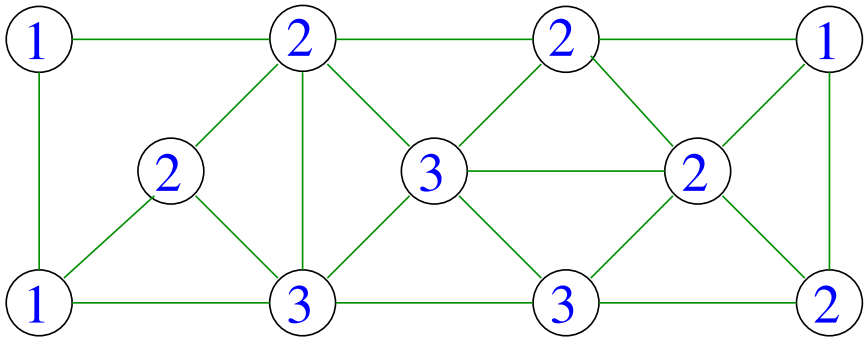
1. A formulação que fizemos do TSP foi para grafos não orientados. Faça uma formulação para o TSP quando as arestas são orientadas (estamos procurando por um circuito hamiltoniano orientado de peso mínimo).
2. Faça uma formulação para encontrar o caminho mínimo de um vértice s e um vértice t, em um grafo com pesos positivos nas arestas usando desigualdades de corte.

Survivable Network Design Problem

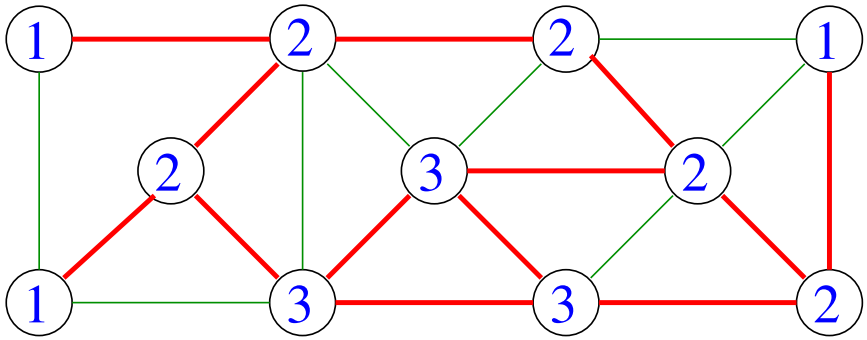
- Neste problema temos vários pontos em uma rede de telecomunicações e alguns pontos que devem ser conectados com requisitos de conectividade.
- Requisitos de conectividade são dados por uma função $f : V \times V \to \mathbb{N}$ que indica o grau de conectividade entre pares de nós.
- Dado dois nós u e v, a função $f(u, v)$ indica a quantidade de rotas alternativas que devem existir entre u e v. Assim, se $f(u, v) = 3$ indica que devem cair no mínimo 3 links na rede para que os nós u e v fiquem desconectados.
- Um subgrafo de G que satisfaz os requisitos de conectividade de f é dita ser uma rede f-conectada de G.
- Para ver mais sobre este problema, veja Mechthild Stoer'92.

Problema SNDP: Dados um grafo $G = (V, E)$, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e e função $f : V \times V \to \mathbb{N}$ encontrar rede f-conectada em G de custo mínimo.

Exemplo: Para simplificar a função f, considere o seguinte grafo onde f é dada como: $f(u, v) = \min\{label(u), label(v)\}$.
Se o vértice u tem label 3 e o vértice v tem label 2, então $f(u, v) = 2$.



Eis uma solução que satisfaz os requisitos de conectividade da função f.



Vamos definir soluções através de variáveis binárias x_{ij} tal que
$x_{ij} = 1$ indica que a aresta ij pertence à solução
$x_{ij} = 0$ indica que a aresta ij não pertence à solução
Estratégia é a mesma do Problema da Floresta de Steiner.
Usaremos desigualdades de corte para garantir f-conectividade.

Formulação:

$$\text{minimize} \sum_{e \in E} c_e x_e$$

$$\text{sujeito a} \begin{cases} \sum_{e \in \delta(S)} x_e \geq K_S & \forall S \subset V, K_S = \max\{f(u,v) : u \in S,\ v \in V \setminus S \\ & \text{(desigualdades de corte)} \\ x_e \in \{0,1\} & \forall e \in E \end{cases}$$

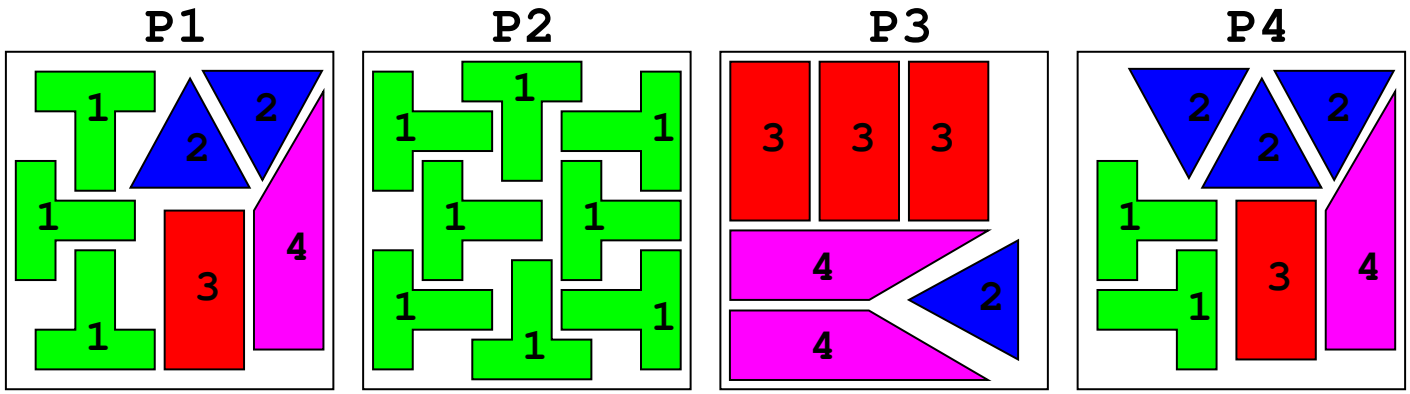
As desigualdades de corte garantem que toda solução inteira deve ter pelo menos os requisitos de conectividade necessários.

Formulações com variáveis inteiras

Problema de Empacotamento

Uma empresa deve vender objetos $O = \{1, 2, \ldots, m\}$, cada objeto i com demanda d_i que devem ser recortados a partir de placas que devem ter uma configuração previamente estabelecida. Há um total de k configurações possíveis e cada configuração j tem a_{ij} items do objeto i. O objetivo é encontrar as quantidades que a empresa deve cortar de cada configuração para suprir a demanda, cortando o menor número de placas possível.

Exemplo de configurações com a disposição dos itens dentro de cada placa.



Ex.: A placa P3 tem 0 itens do objeto 1, 1 item do objeto 2, 3 itens do objeto 3 e 2 itens do objeto 4.

Formulação do problema de empacotamento

Vamos definir soluções através de variáveis inteiras $x_j \geq 0$, que dá a quantidade de placas na configuração j que devemos cortar.
Restrições para cada objeto i:

- As placas a serem cortadas devem suprir a demanda do objeto d_i.
- A quantidade de placas na configuração j é x_j.
- Cada configuração j tem a_{ij} itens do tipo i.

Assim, para satisfazer a demanda d_i, devemos impor que

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{ik}x_k \geq d_i$$

Considere as configurações apresentadas na figura anterior. Suponha que $d_1 = 950$, $d_2 = 1150$, $d_3 = 495$ e $d_4 = 450$. A formulação ficaria a seguinte:

Formulação:

$$
\begin{array}{ll}
\text{minimize} & x_1 + x_2 + x_3 + x_4 \\
\text{sujeito a} & \left\{ \begin{array}{llllll}
3x_1 & +8x_2 & & +2x_4 & \geq & 950 \\
2x_1 & & +1x_3 & +3x_4 & \geq & 1150 \\
1x_1 & & +3x_3 & +1x_4 & \geq & 495 \\
1x_1 & & +2x_3 & +1x_4 & \geq & 450 \\
x_i \in \mathbb{Z}^* & & & & &
\end{array} \right.
\end{array}
$$

Cada linha i contém os dados de um objeto i e cada coluna j contém os dados da configuração j.

Formulação:

$$
\begin{array}{ll}
\text{minimize} & x_1 + x_2 + \cdots + x_k \\
\text{sujeito a} & \left\{ \begin{array}{lcl}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1k}x_k & \geq & d_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2k}x_k & \geq & d_2 \\
\vdots & \vdots & \vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nk}x_k & \geq & d_n \\
x_i \in \mathbb{Z}^* &&
\end{array} \right.
\end{array}
$$

Outras aplicações: Corte de barras, alocação de comerciais de TV, alocação em páginas de memória, corte de placas (vidro, madeira, chapas, tecido, espuma, etc), empacotamento em contêineres, etc.

Truques de modelagem

- **Desigualdades excludentes**
  Deve valer apenas uma entre duas desigualdades:
  ou $a'x \leq \alpha'$ ou $a''x \leq \alpha''$ devem valer, não ambas.
  Use nova variável binária Δ com valor 0 se vale a primeira desigualdade e 1 caso a segunda deva ocorrer.

$$\begin{array}{rrcl} a'x & -M\cdot\Delta & \leq & \alpha' \\ a''x & -M\cdot(1-\Delta) & \leq & \alpha'' \\ & \Delta & \in & \{0,1\} \end{array}$$

  onde M é um termo suficientemente grande.

Obs.: Programas que apresentam valores de M grande podem provocar soluções “muito fracionárias”.

- **Alternativas no lado direito de igualdades**
  Se deve valer a igualdade $ax = \alpha$ onde $\alpha \in \{\alpha_1, \ldots, \alpha_k\}$, usamos novas variáveis binárias $\Delta_1, \ldots, \Delta_k$ onde $\Delta_i = 1$ se e somente se a igualdade satisfeita é $ax = \alpha_i$.

$$ax = \sum_{i=1}^{k} \alpha_i \Delta_i \qquad \sum_{i=1}^{k} \Delta_i = 1$$
$$\Delta_i \in \{0, 1\}$$

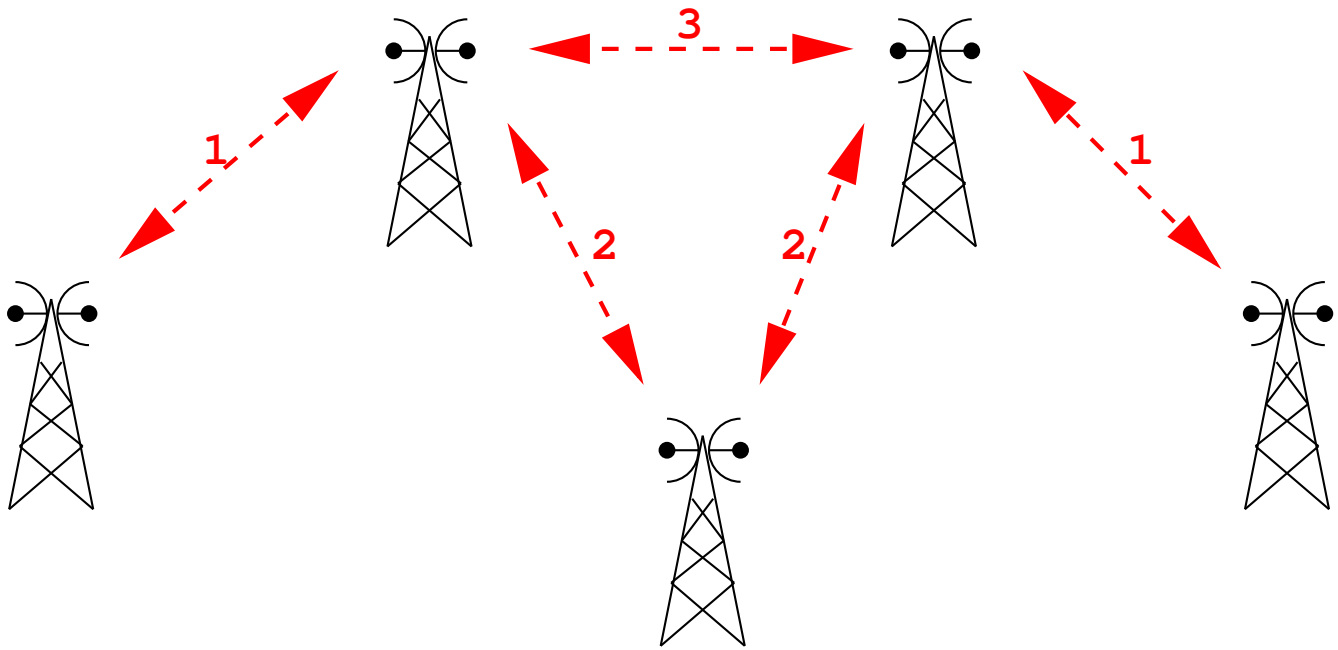
- **Penalidades em desigualdades**
  As vezes permitimos que uma desigualdade $ax \leq \alpha$ seja violada, mas quanto maior a violação, penalizamos com fator P:

$$\text{minimize } \ldots + yP$$

$$ax \leq \alpha + y$$

$$y \geq 0$$

Problema ATRIBUIÇÃO DE FREQÜÊNCIAS *(Radio link frequency assignment problem)*: Dados conjunto de antenas $A = \{a_1, a_2, \ldots, a_n\}$, conjunto de freqüências $F = \{1, \ldots, K\}$, K é uma variável, e uma função distância $d : A \times A \to \mathbb{N}$, encontrar uma atribuição de freqüências para as antenas $f : A \to F$ tal que a distância das freqüências atribuídas para as antenas a_i e a_j é pelo menos $d(a_i, a_j)$. O objetivo é encontrar uma atribuição de freqüências viável que minimize o valor de K.



Formulação:

- Variáveis f_i indicando a freqüência que iremos atribuir à antena i;
- todas as freqüências devem ocorrer no intervalo $\{1, \ldots, K\}$:

$$1 \leq f_i \leq K, \quad \forall i \in A.$$

- O objetivo é minimizar K.
- A atribuição deve satisfazer a função de distância:

$$|f_i - f_j| \geq d(i,j), \qquad i \in A \quad e \quad j \in A$$

Ops, módulo não é função linear. Vamos transformar para restrições lineares.
Vamos usar uma variável binária Δ_{ij} para indicar as alternativas do módu

- se $\Delta_{ij} = 0$ então deve ocorrer $f_i - f_j \geq d(i,j)$
- e• se $\Delta_{ij} = 1$ então deve ocorrer $f_j - f_i \geq d(i,j)$.

Podemos trocar a restrição não linear

$$|f_i - f_j| \geq d(i,j)$$

pelas seguintes restrições lineares

$$\begin{array}{rrcl} f_i - f_j & +M\cdot\Delta_{ij} & \geq & d(i,j) \\ f_j - f_i & +M\cdot(1-\Delta_{ij}) & \geq & d(i,j) \\ & \Delta_{ij} & \in & \{0,1\} \end{array}$$

onde M é um número suficientemente grande.
Se $\Delta_{ij} = 0$, a primeira restrição nos dá

$$f_i - f_j \geq d(i,j) \text{ e conseqüentemente } f_i \geq f_j$$

a segunda restrição é sempre válida:

$$f_j - f_i + M \geq d(i,j)$$

o que é sempre verdade se M for suficientemente grande.
Analise análoga pode ser feita quando $\Delta_{ij} = 1$.

Obtemos a seguinte formulação inteira para o problema de atribuição de freqüências:

$$
\begin{array}{ll}
\text{minimize} & K \\
\text{sujeito a} & \left\{
\begin{array}{rcll}
f_i - f_j + M\cdot\Delta_{ij} & \geq & d(i,j) & \forall i \neq j, i \in A, j \in A \\
f_j - f_i + M\cdot(1-\Delta_{ij}) & \geq & d(i,j) & \forall i \neq j, i \in A, j \in A \\
\Delta_{ij} & \in & \{0,1\} & \forall i \neq j, i \in A, j \in A \\
f_i & \geq & 1 & \forall i \in A \\
f_i & \leq & K & \forall i \in A \\
f_i & \in & \mathbb{Z} & \forall i \in A \\
K & \in & \mathbb{Z} &
\end{array}
\right.
\end{array}
$$

Obs.: A formulação deste problema foi simplificada. Algumas vezes há custos (interferência) envolvida na atribuição que dependem da distância entre freqüências para cada par de antenas. Algumas antenas tem restrições de freqüências, por estarem próximas a antenas que estão fora do controle da atribuição (e.g., pertencem a outras empresas). Para ver mais sobre este problema, veja Borndörfer, Eisenblätter, Grötschel, Martin'96 (ZIB).

Problema ESCALONAMENTO COM PRECEDÊNCIA: Dados uma lista de tarefas $L = \{1, \ldots, n\}$, cada uma com tempo inteiro p_i, uma ordem parcial $\prec$ entre tarefas e uma função de prioridade $w : L \to \mathbb{R}^+$,

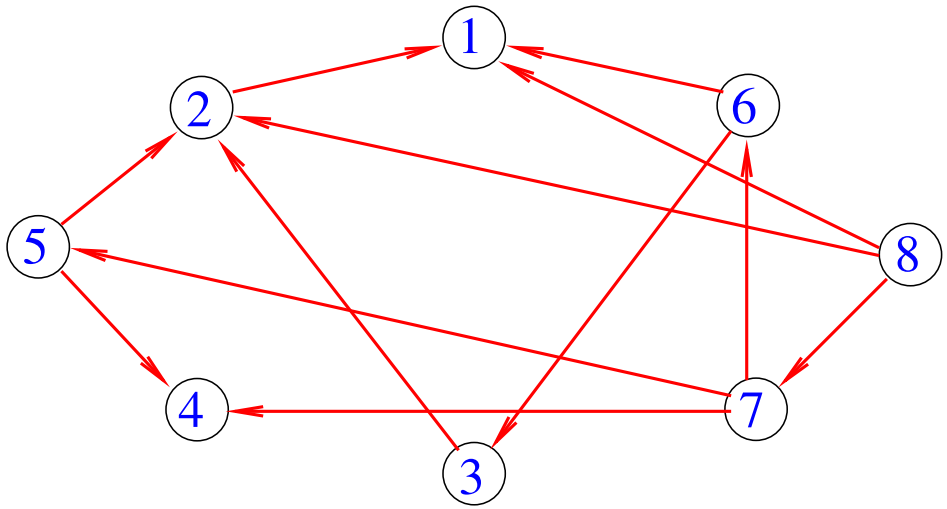
encontrar um escalonamento de L, obedecendo precedência, em um processador tal que $\sum_i w_i f_i$ é mínimo, onde f_i é o tempo que a tarefa i é finalizada.

Um escalonamento obedece precedências se para todo par de tarefas (i, j) para o qual vale $i \prec j$ então a tarefa i termina antes da tarefa j começar. A execução das tarefas não podem ser feitas por partes.

Teorema: *O* ESCALONAMENTO COM PRECEDÊNCIA *é um problema NP-difícil.*

Problemas de escalonamento em geral: Escalonamento de processos em máquinas paralelas, escalonamento de pessoal em turnos de trabalho, etc

Exemplo de precedência através de um grafo orientado:
($i \leftarrow j$ significa que i deve ser executado antes de j)



Exemplos de escalonamentos viáveis:

1 - 2 - 3 - 4 - 5 - 6 - 7 - 8
4 - 1 - 2 - 5 - 3 - 6 - 7 - 8
1 - 4 - 2 - 3 - 6 - 5 - 7 - 8

Vamos ver duas formulações para este problema

Formulação 1:

- s_i variável inteira indicando o tempo de início da tarefa i
- f_i variável inteira indicando o tempo que a tarefa i é finalizada
- Δ_{ij} variável binária indicando se a tarefa i vem antes de j
  Δ_{ij} tem valor 1 se tarefa i vem antes de j, 0 caso contrário.

Função objetivo: Minimizar o tempo de finalização com prioridades:

$$\text{minimize } \sum_{i=1}^{n} w_i f_i$$

Tempos não negativos:

$$s_i \geq 0, \;\; f_i \geq 0, \text{ para } 1 \leq i \leq n$$

Término é tempo de início + tempo de processamento: $s_i + p_i = f_i$.

$$f_i - s_i = p_i$$

Formulação 1:

Quando $i \prec j$

$$f_i \leq s_j$$

Quando não há restrição entre i e j,

$$\begin{array}{rrcl} f_i & -M(1-\Delta_{ij}) & \leq & s_j \\ f_j & -M\Delta_{ij} & \leq & s_i \end{array}$$

onde M é um valor suficientemente grande.

Formulação 1:

$$\text{minimize} \quad \sum_{i=1}^{n} w_i f_i$$

$$\text{sujeito a} \begin{cases} -s_i + f_i = p_i & \forall i \in [n] \\ -s_i + f_i - M(1-\Delta_{ij}) \leq 0 & \forall i < j : \ ij \notin \mathcal{P} \text{ e } ji \notin \mathcal{P} \\ -s_i + f_j - M\Delta_{ij} \leq 0 & \forall i < j : \ ij \notin \mathcal{P} \text{ e } ji \notin \mathcal{P} \\ -s_j + f_i \leq 0 & \forall ij : i \prec j \\ s_i \geq 0 & \forall i \in [n] \\ \Delta_{ij} \in \{0,1\} & \forall ij \in [n] \times [n] \end{cases}$$

onde $[n] = \{1, \ldots, n\}$ e M é um valor suficientemente grande.

Formulação 2:

Vamos supor que o tempo é medido em unidades inteiras.
e é possível terminar todas as tarefas em um tempo máximo T.

- f_j tempo que a tarefa j terminou.
- x_{jt} vale 1 se a tarefa j termina no tempo t, e vale 0 caso contrário

Uma tarefa só pode terminar em um determinado tempo:

$$\sum_{t=1}^{T} x_{jt} = 1, \text{ para } 1 \leq j \leq n$$

$x_{jt} = 1$ se e somente se $f_j = t$:

$$f_j = \sum_{t=1}^{T} t\, x_{jt}, \text{ para } 1 \leq j \leq n$$

Função objetivo: Minimizar o tempo de finalização com prioridades:

$$\text{minimize } \sum_{j=1}^{n} w_j f_j$$

Nenhuma tarefa pode terminar antes do seu tempo de processamento.

$$x_{jt} = 0, \text{ para } t < p_j \text{ e } 1 \leq j \leq n$$

Obs.: Uma estratégia nesta formulação é considerar o seguinte:

$$\sum_{s=t_i}^{t_f} x_{js} = 1$$

equivale a dizer que j terminou o processamento no tempo $[t_i, \ldots, t_f]$.
Se $j \prec k$ então j deve terminar pelo menos p_k unidades de tempo antes de k terminar.

$$\sum_{s=1}^{t} x_{js} \geq \sum_{s=1}^{t+p_k} x_{ks}, \text{ para } j \prec k \text{ e } t = 1, \ldots, T - p_k$$

Em cada tempo t, deve haver apenas um job sendo executado

$$\sum_{j=1}^{n} \sum_{s=t}^{\min\{t+p_j-1, T\}} x_{js} \leq 1, \text{ para } t = 1, \ldots, T$$

Note que se $\sum_{s=t}^{\min\{t+p_j-1, T\}} x_{js} = 1$ significa que o tempo t está sendo usado pela tarefa j.

$$\text{minimize} \quad \sum_{j=1}^{n} w_j f_j$$

$$\text{sujeito a} \begin{cases} \displaystyle\sum_{t=1}^{T} x_{jt} = 1 & \forall 1 \le j \le n \\ f_j - \displaystyle\sum_{t=1}^{T} t\, x_{jt} = 0 & \forall 1 \le j \le n \\ x_{jt} = 0 & \forall t < p_j \text{ e } 1 \le j \le n \\ \displaystyle\sum_{s=1}^{t} x_{js} - \sum_{s=1}^{t+p_k} x_{ks} \ge 0 & \forall ij : j \prec k \text{ e } t = 1, \ldots, T - p_k \\ \displaystyle\sum_{j=1}^{n} \sum_{s=t}^{\min\{t+p_j-1, T\}} x_{js} \le 1 & t = 1, \ldots, T \\ x_{jt} \in \{0, 1\} & \forall 1 \le t \le T \text{ e } 1 \le j \le n \end{cases}$$

Apesar de mais variáveis e mais complicada, esta formulação tem apresentado melhores resultados que a primeira formulação

Exercícios:

- Em uma formulação linear inteira, uma variável x pode assumir valor 0 ou valores acima de uma constante K. Como você restringe para que isto ocorra ?
- Seu programa linear inteiro deve ter restrições $ax \leq \alpha_1, \ \ldots \ ax \leq \alpha_m$, mas no máximo t, $1 \leq t \leq m$ restrições devem realmente ser satisfeitas. Como você pode formular isto ?
- Faça uma formulação alternativa para o problema de atribuição de freqüências, usando variáveis binárias x_{if} com valor 1 se e somente se a antena i recebeu a freqüência f (suponha que a maior freqüência é K).

Programa Inteiro e Programa Relaxado

Seja P um poliedro (relaxado) e I o conjunto de pontos inteiros em P seja P_I o correspondente poliedro inteiro em P (i.e., $P_I := conv(I)$).
Queremos otimizar em P_I, mas em geral só temos P.
Informações de P e P_I:

- As soluções de P_I e P podem estar muito próximas.
- Todos os pontos de P_I pertencem a P.
- Se a função objetivo é de minimização, a solução ótima de P é menor ou igual à solução ótima de P_I.
- Se a função objetivo é de maximização, a solução ótima de P é maior ou igual à solução ótima de P_I.

Resolvendo PLI por Arredondamento

Este método consiste em

- modelar o problema como problema de programação linear inteira
- relaxar a formulação para programação linear
- resolver o programa linear
- arredondar as variáveis para cima/baixo, de acordo com o problema.

Vejamos este método aplicado ao exemplo do problema do empacotame

Formulação em PLI:

$$
\begin{array}{ll}
\text{minimize} & x_1 + x_2 + x_3 + x_4 \\
\text{sujeito a} & \left\{ \begin{array}{llllll}
3x_1 & +8x_2 & & +2x_4 & \geq & 950 \\
2x_1 & & +1x_3 & +3x_4 & \geq & 1150 \\
1x_1 & & +3x_3 & +1x_4 & \geq & 495 \\
1x_1 & & +2x_3 & +1x_4 & \geq & 450 \\
x_i \geq 0, & & x_i \in \mathbb{Z} & & &
\end{array} \right.
\end{array}
$$

Relaxação do PLI:

$$
\begin{array}{ll}
\text{minimize} & x_1 + x_2 + x_3 + x_4 \\
\text{sujeito a} & \left\{ \begin{array}{llllll}
3x_1 & +8x_2 & & +2x_4 & \geq & 950 \\
2x_1 & & +1x_3 & +3x_4 & \geq & 1150 \\
1x_1 & & +3x_3 & +1x_4 & \geq & 495 \\
1x_1 & & +2x_3 & +1x_4 & \geq & 450 \\
x_i \geq 0 & & & & &
\end{array} \right.
\end{array}
$$

Resolvendo este programa linear, obtemos uma solução de valor 437,656 e valores $x_1 = 0$, $x_2 = 26{,}406$, $x_3 = 41{,}875$ e $x_4 = 369{,}375$.
O número de placas é inteiro $\Rightarrow$ qualquer solução ótima usa pelo menos 438 placas.
Arredondando as variáveis fracionárias para cima, obtemos uma solução de valor $x_1 = 0$, $x_2 = 27$, $x_3 = 42$ e $x_4 = 370$, que nos dão uma solução de 439 placas.
Assim, se a nossa solução não for ótima, estamos usando uma placa a mais que a solução ótima.

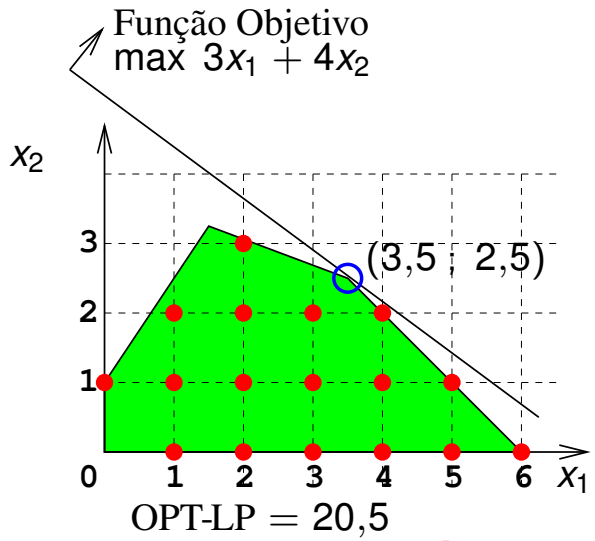
Branch & Bound e Programação Linear Inteira

- modelar o problema como problema de programação linear inteira
- relaxar a formulação para programação linear
- enumerar o espaço de soluções através do método branch & bound. Cada nó da árvore representa um programa linear. Cada programa linear (nó) é ramificado enquanto houver valores fracionários em sua solução.

Considere o programa linear inteiro:

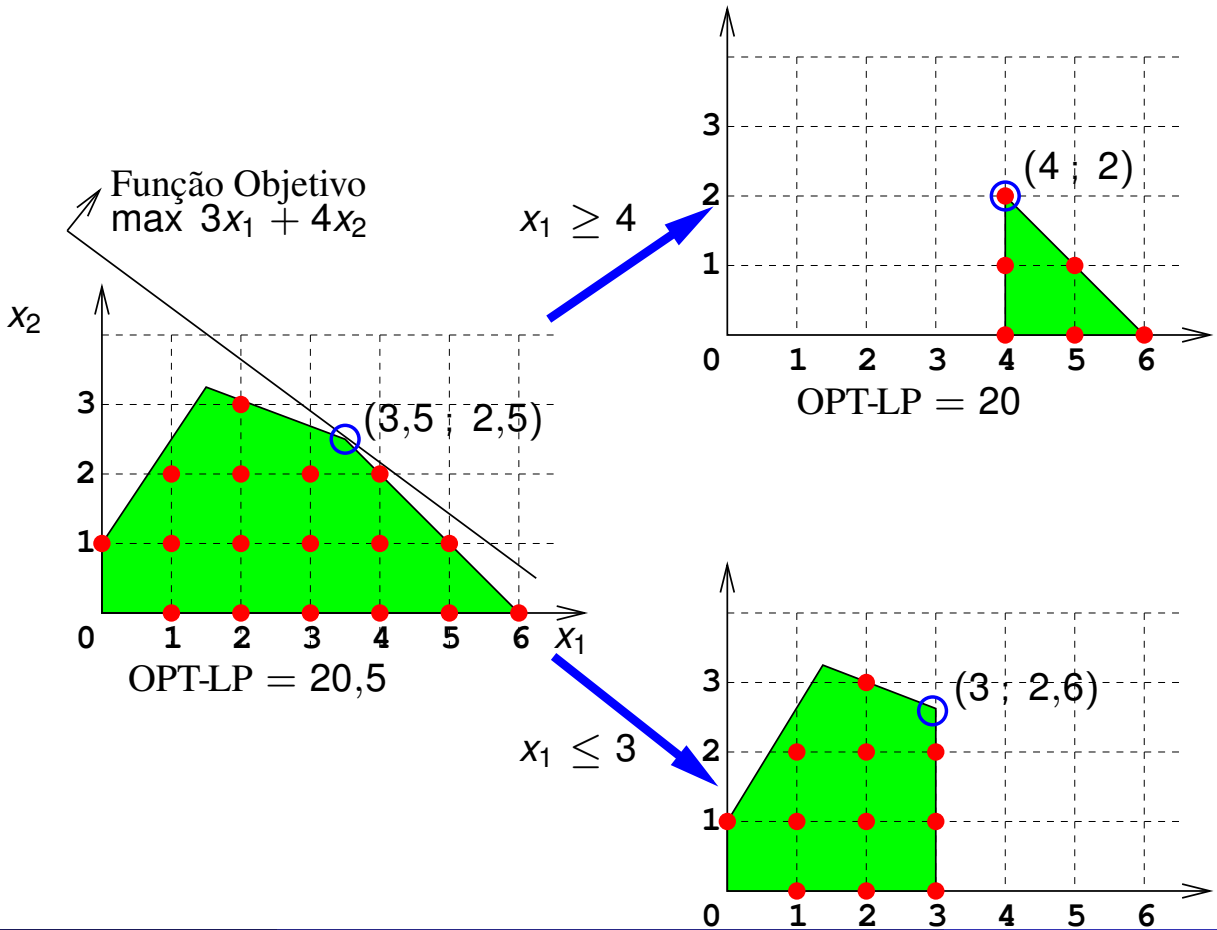
$$\begin{array}{ll} \text{maximize} & 3x_1 + 4x_2 \\ \text{sujeito a} & \begin{cases} -3x_1 + 2x_2 \leq 2 \\ x_1 + 3x_2 \leq 11 \\ x_1 + x_2 \leq 6 \\ x_1 \geq 0, \quad x_2 \geq 0 \\ x_i \in \mathbb{Z}^* \end{cases} \end{array}$$

Representação gráfica do programa relaxado e os pontos inteiros:

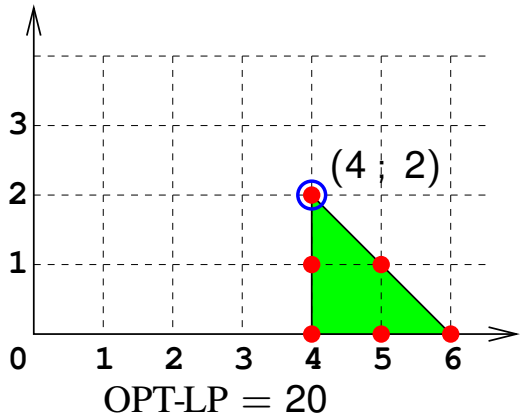


Vamos resolver o programa inteiro através do Branch (& Bound)

A primeira solução ótima fracionária corresponde ao ponto ($x_1 = 3{,}5\ ;\ x_2 = 2{,}5$). Escolhendo a variável x_1 (que tem valor fracionário 3,5) para fazer branching, separamos o problema em duas partes. Uma inserindo a restrição $x_1 \leq 3$ e outra inserindo a restrição $x_1 \geq 4$. Os dois subproblemas estão representados a seguir:

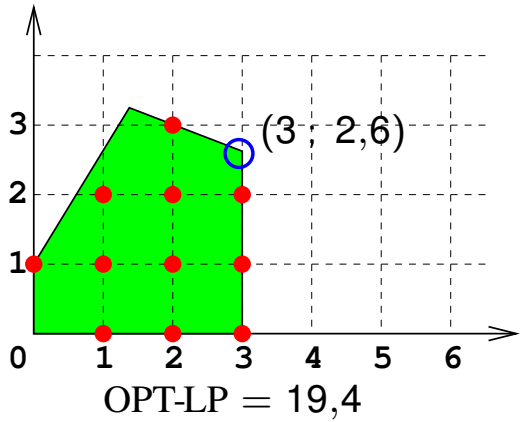


Um dos programas já tem solução inteira e não precisamos continuar sua ramificação:



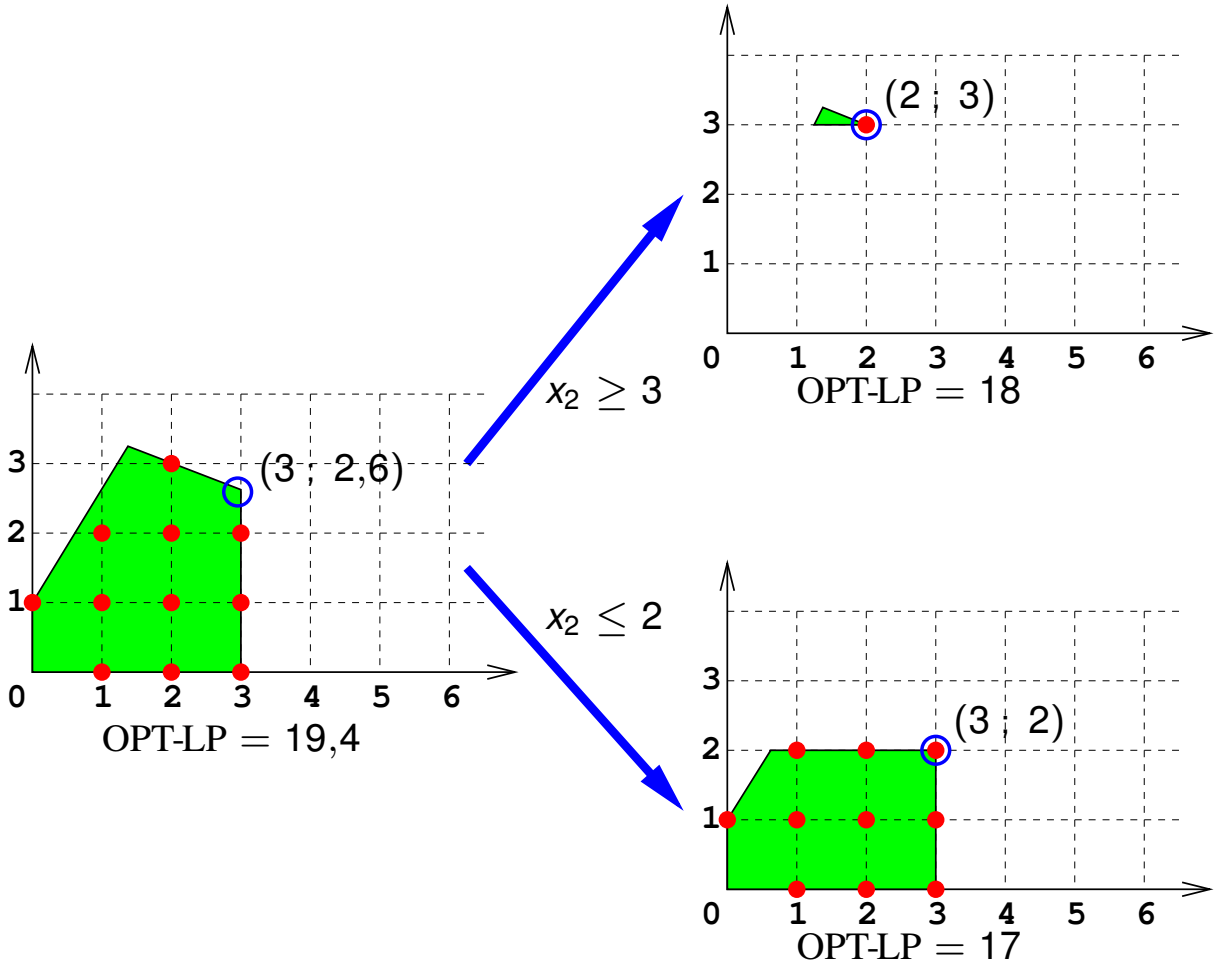
OPT-LP $= 20$

O outro programa tem solução ótima em (3 ; 2,6) e portanto fracionário na variável x_2. Ainda é necessário ramificar este nó:



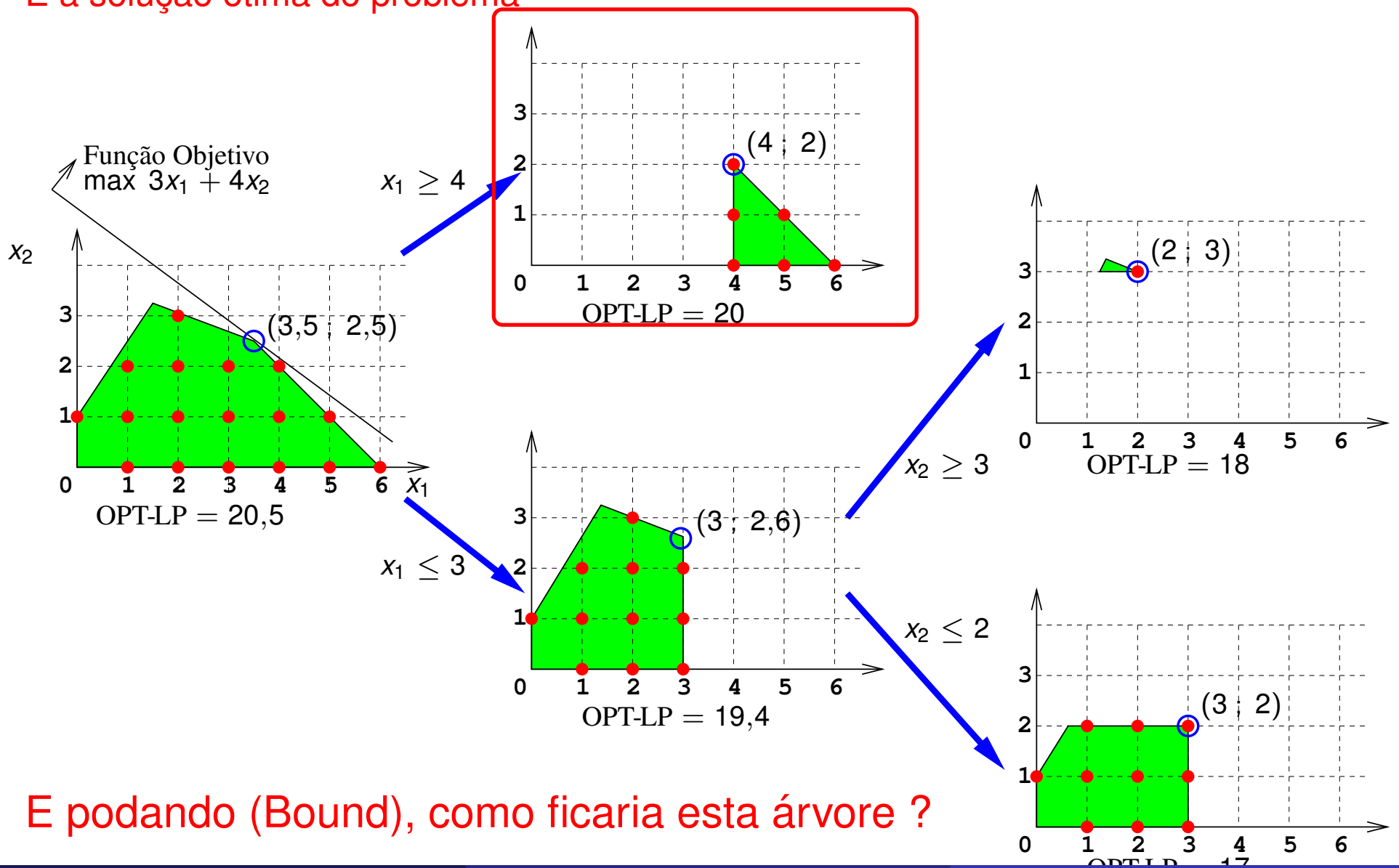
OPT-LP $= 19{,}4$

Ramificando em x_2, para os casos $x_2 \leq 2$ e $x_2 \geq 3$, obtemos os seguintes programas



Árvore completa de Branch (sem Bound).

E a solução ótima do problema



E podando (Bound), como ficaria esta árvore ?

Estratégias comuns envolvidas na árvore de B&B

- **Escolhendo o nó a ramificar**
  Usamos o nó que tem a maior distância entre o limitante inferior e superior.
  Por exemplo, em um problema de minimização, pegue o nó que tem o menor limite inferior.
  Isto permite diminuir a diferença entre limitantes superior e inferior.
- **Escolha da variável para ramificar:**
  - Variável “mais fracionária”: Parte fracionária mais próxima de 0,5
  - Variável “menos fracionária”: Parte fracionária mais distante de 0,5

Estratégias comuns envolvidas na árvore de B&B

- **Ramificação por restrição:**
  Encontre uma restrição válida $ax \leq b$ e divida em duas partes:
  1. Um ramo tem inserido a desigualdade $ax \leq b'$ e
  2. outro ramo tem inserido a desigualdade $ax \geq b''$.

  Ex.: Encontre uma desigualdade do TSP do tipo $x(\delta(S)) \geq 2$ e
  1. coloque a restrição $x(\delta(S)) = 2$ em um ramo e
  2. coloque a restrição $x(\delta(S)) \geq 3$ em outro ramo.

  (naturalmente há um esforço para encontrar tal desigualdade)
- **Usando apenas um programa linear:**
  Na maioria das vezes usamos apenas um programa linear. Resolva o programa linear associado a um nó, acertando previamente os limitantes das variáveis daquele nó. A resolução de programas lineares anteriormente resolvidos com pequenas modificações é em geral mais rápida.

Estratégias comuns envolvidas na árvore de B&B

- **Representando a árvore de Branch & Bound:**
  Algumas vezes não precisamos armazenar a árvore, mas uma lista dos nós ativos.
  Cada vez que escolhemos um nó ativo para ramificar, removemos este do conjunto e inserimos seus ramos (novos nós).
  Estrutura de dados comum para armazenar nós: Fila de prioridade
- **Guarde a melhor solução viável**
- **Poda da árvore:**
  Use o valor da melhor solução encontrada combinada com estratégias de limitantes superiores para podar ramos não promissores.
  Boas estratégias de poda podem evitar crescimento rápido da árvore.

- **Percorrendo a árvore de B&B**
  - Se não temos solução viável: aplicar busca em profundidade por ramos mais promissores
  - Se temos solução viável: misture escolha do melhor nó com busca em profundidade
- **Branch & Bound para programas com variáveis em $\{0, 1\}$:**
  Ramos da enumeração consistem em fixar variáveis para 0 ou 1.
- **Fixação de variáveis por implicações lógicas:**
  A fixação do valor de algumas variáveis pode levar a fixar outras.

  Considere o problema do TSP resolvido através de B & B com LP.

  Removendo as arestas fixadas em 0 e contraindo arestas fixadas em 1, podemos obter um grafo tal que:

  - se houver vértice de grau dois, podemos incluir as arestas incidentes na solução do nó.

  - se houver uma aresta cuja remoção desconecta o grafo, podemos podar o ramo deste nó.

BRANCH-BOUND-MINIMIZAÇÃO-SIMPLIFICADO (P^o) % PL relaxado P^o

1 LB$\leftarrow$ solução ótima de P^o (halt se P^0 é inviável)
2 $x^* \leftarrow$ Heurística sobre P^o ($\emptyset$ se não obteve solução viável/não há)
3 UB$\leftarrow \text{val}(x^*)$ (onde $\text{val}(\emptyset) = +\infty$)
4 Ativos$\leftarrow \{P^o\}$
5 enquanto Ativos$\neq \emptyset$ faça
6 escolha $P \in$ Ativos
7 Ativos $\leftarrow$ Ativos $\setminus \{P\}$
8 se P é viável, % se inviável, ramo é podado
9 $x \leftarrow$ solução ótima de P
10 se $\text{val}(x) > LB$ então atualiza LB (se necessário)
11 se $\text{val}(x) < UB$ % se $\text{val}(x) \geq UB$, ramo é podado
12 se x é inteiro então
13 $UB \leftarrow \text{val}(x)$
14 $x^* \leftarrow x$
15 senão
16 crie dois subproblemas P' e P'' a partir de P
17 Ativos$\leftarrow$Ativos$\cup \{P' , P''\}$
18 devolva x^*

Método de Planos de Corte

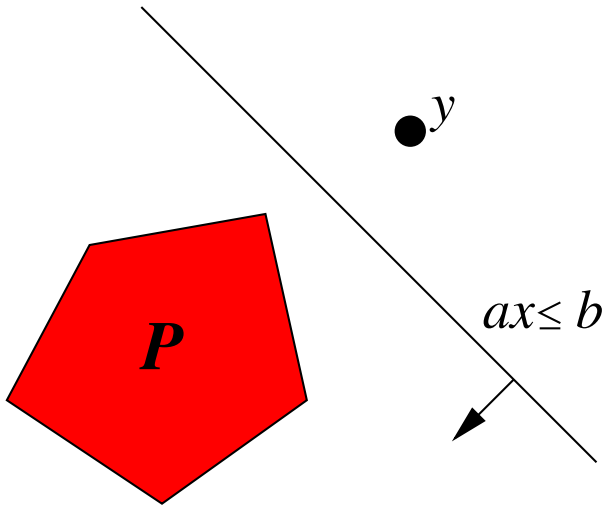
Seja (P, c) um programa linear, onde P é um poliedro e c é a função objetivo.
Suponha que P é descrito por número grande de desigualdades.
Ainda assim, podemos obter uma solução ótima para (P, c).
Estratégia:

1. Não usar P, mas começar com um poliedro inicial Q (descrito por poucas desigualdades) que contém P
2. Repetidamente encontrar uma solução ótima y de Q e caso $y \notin P$, adicionamos a Q uma desigualdade válida (chamada de plano de corte) que separa (corta) y de P sem perder nenhuma solução de P.
3. Parar quando encontrar uma solução ótima de P.

Precisamos repetir o passo 2 muitas vezes ?

Otimização × Separação

Def.: Problema da Separação: Seja $P \subseteq \mathbb{R}^n$ um conjunto convexo e $y \in \mathbb{R}^n$, determinar se $y \in P$, caso contrário, encontrar desigualdade $ax \leq b$ tal que $P \subseteq \{x : ax \leq b\}$ e $ay > b$.



Teorema: *(Grötschel, Lovász, Schrijver'81) O problema de otimização de um programa linear pode ser resolvido em tempo polinomial se e somente se o problema da separação para o poliedro do programa linear pode ser resolvido em tempo polinomial.*

ALGORITMO PLANOS DE CORTE(P, c) Dantzig, Fulkerson e Johnson'54

P poliedro (não necessariamente explícito),

c função objetivo

1 $Q \leftarrow$ {Poliedro inicial}

2 $y \leftarrow$OPT-LP(Q, c)

3 enquanto y pode ser separado de Q faça

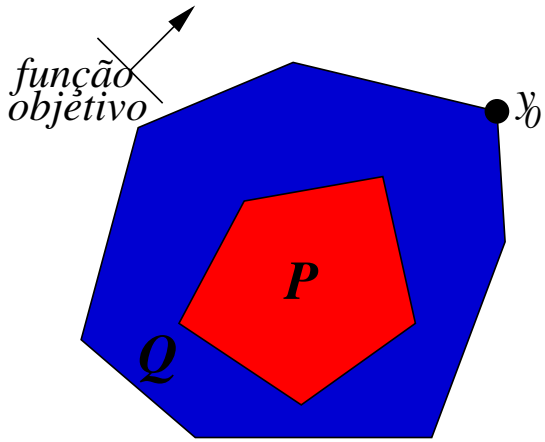
4 seja $ax \leq b$ desigualdade de P que separa y

5 $Q \leftarrow Q \cap \{x : ax \leq b\}$

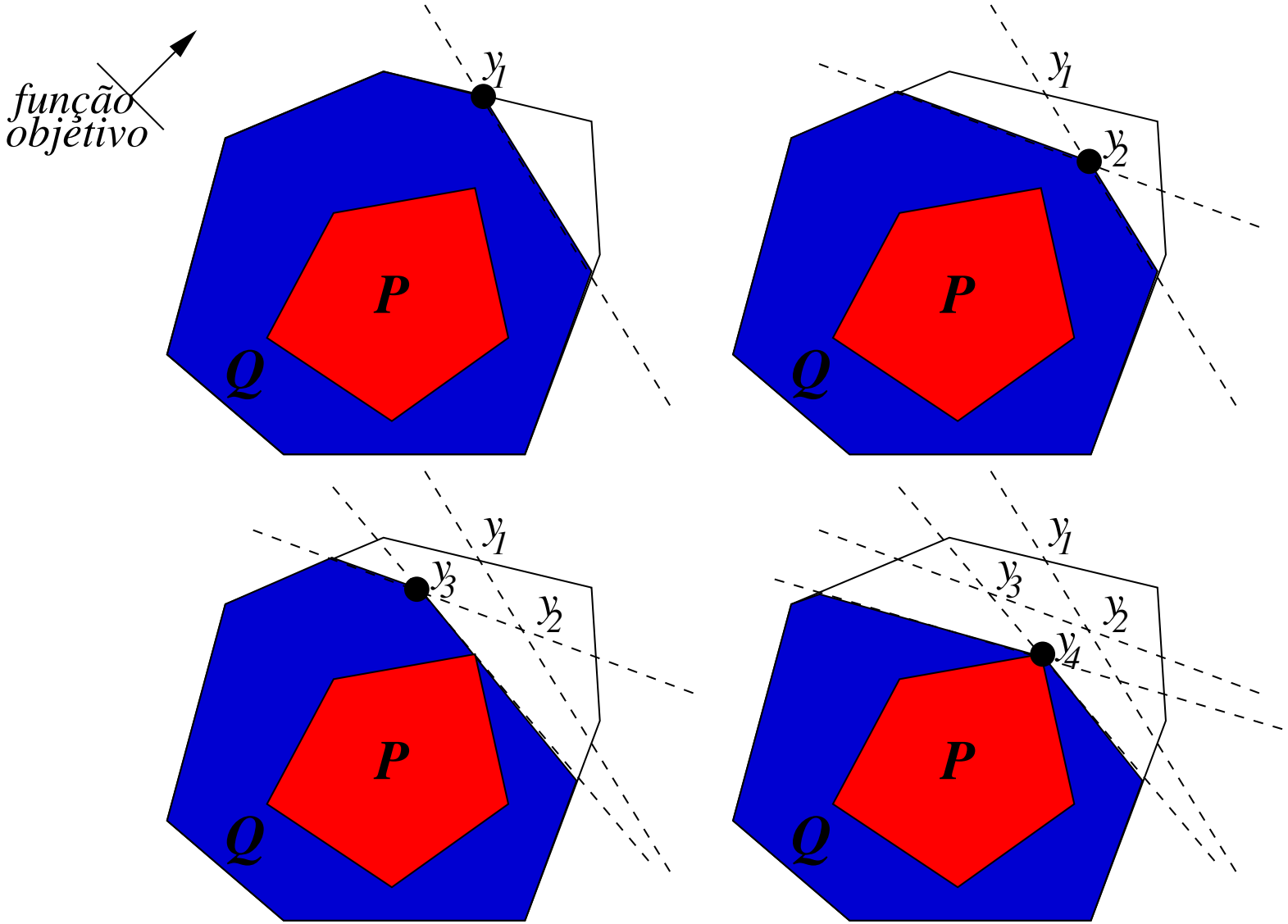
6 $y \leftarrow$OPT-LP(Q, c)

7 se $Q = \emptyset$ retorne $\emptyset$

8 senão retorne y,



onde OPT-LP(Q, c) é a solução ótima do programa linear (Q, c)



Que tipo de desigualdade são as melhores para serem inseridas ?

Planos de corte e Conectividade

Muitos problemas vistos envolvem conectividade:

- Floresta de Steiner
- Caixeiro Viajante
- Survivable Network Design

Por exemplo, no TSP, uma das desigualdades válidas foi a seguinte:

$$\sum_{e \in \delta(S)} x_e \geq 2, \qquad \forall \emptyset \neq S \subset V$$

I.e., para todo conjunto de vértices S, $\emptyset \neq S \subset V$, o número de arestas escolhidas na solução (arestas e com $x_e = 1$) que pertencem ao corte $\delta(S)$ deve ser pelo menos 2.
Como testar se um ponto x satisfaz todas as desigualdades de corte acima ?

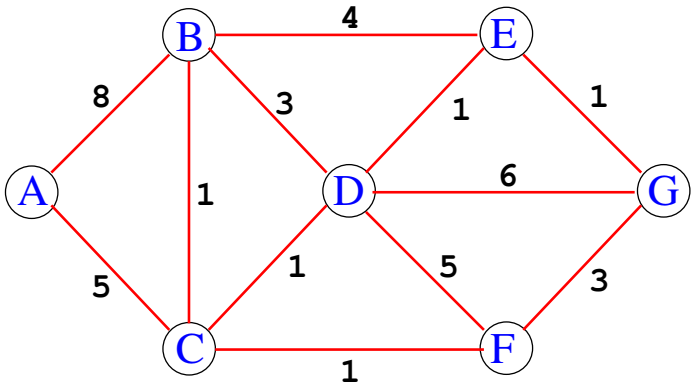
Para testar se dois vértices s e t estão separados por um corte $(S, \overline{S})$ que viola a desigualdade de corte, i.e., está ocorrendo

$$\sum_{e \in \delta(S)} x_e < 2 \quad s \in S, \quad t \in \overline{S}$$

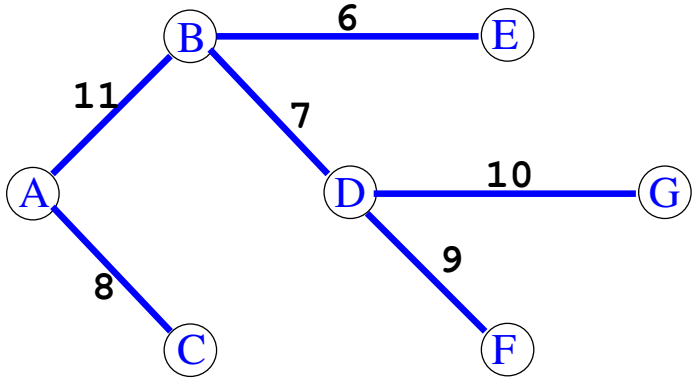
podemos executar o algoritmo para st-corte mínimo.
Note que o algoritmo deve ser executado não para o grafo original, mas onde cada aresta e tem capacidade x_e.
Se o valor do st-corte mínimo for menor que 2, então o corte gerado viola a desigualdade de corte e podemos inseri-la como desigualdade válida.

- Estratégia Direta:
  Testar para todos os pares de vértices: $O(n^2)$ execuções do st-corte mínimo.
- Árvore de Cortes de Gomory-Hu (Gomory, Hu'61):
  Todos os cortes representados por uma árvore usando $n - 1$ execuções do st-corte mínimo

Árvore de Cortes de Gomory-Hu:

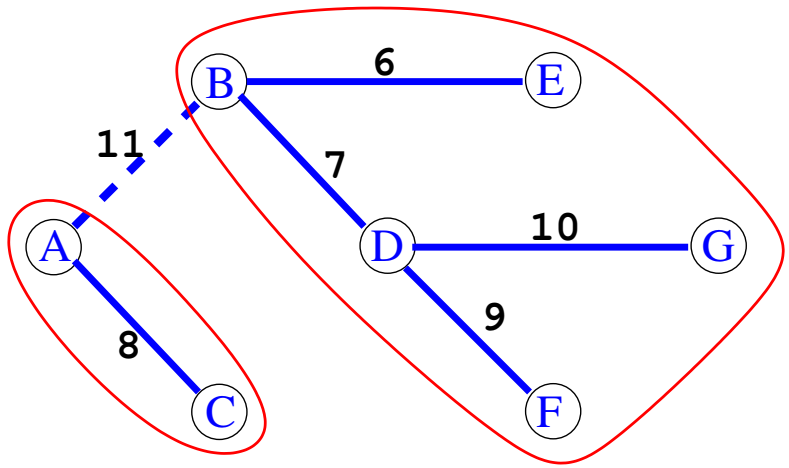


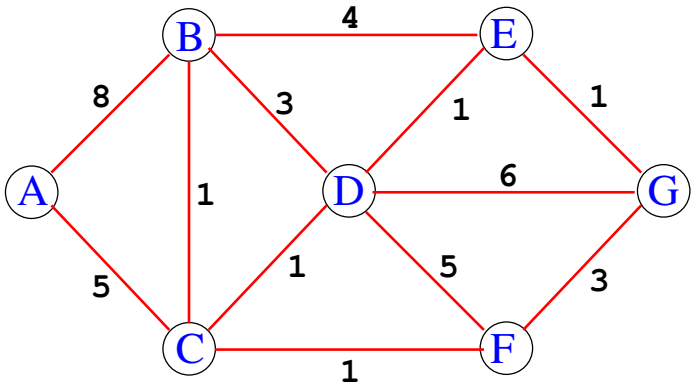
Grafo $G = (V, E)$



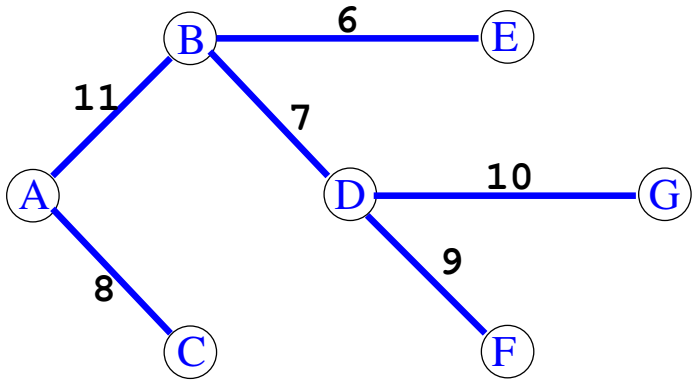
Árvore T de Cortes de Gomory-Hu de G

Cada aresta na árvore de cortes de Gomory-Hu representa um corte.
A capacidade do corte é o peso da aresta em T.
Exemplo do corte de capacidade 11 representado pela aresta (A, B) em T



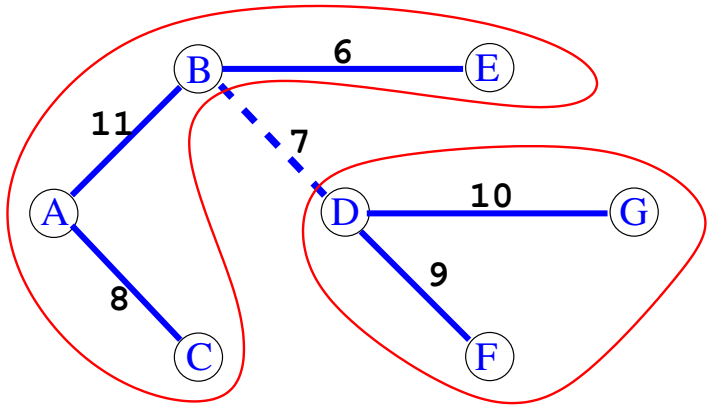


Grafo $G = (V, E)$



Árvore T de Cortes de Gomory-Hu de G

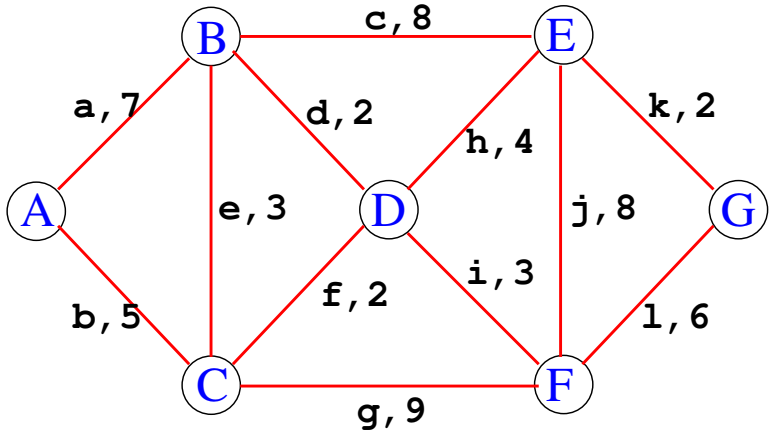
Corte mínimo que separa u e v em G é dado pelas componentes obtidas da remoção da aresta de capacidade mínima no caminho entre u e v em T.



Exemplo de corte mínimo que separa os vértices A e G (7 é o mínimo em $\{11, 7, 10\}$).

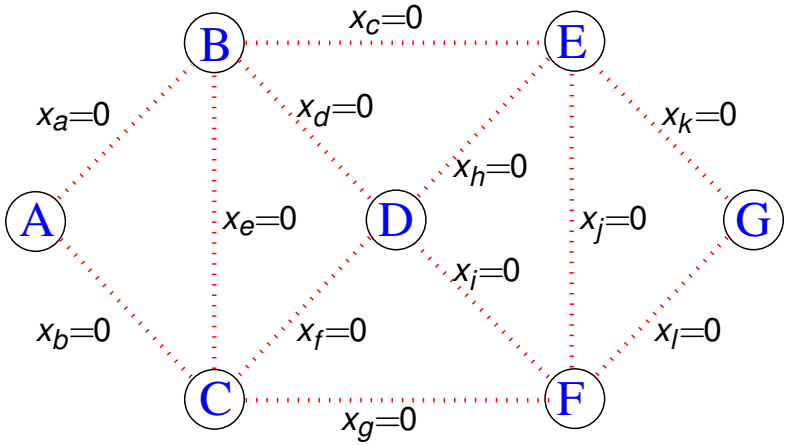
Separação para o TSP

Vamos ver um exemplo de como se comporta a inserção de desigualdades para o TSP para o seguinte grafo:



Grafo com nome de arestas e seus custos

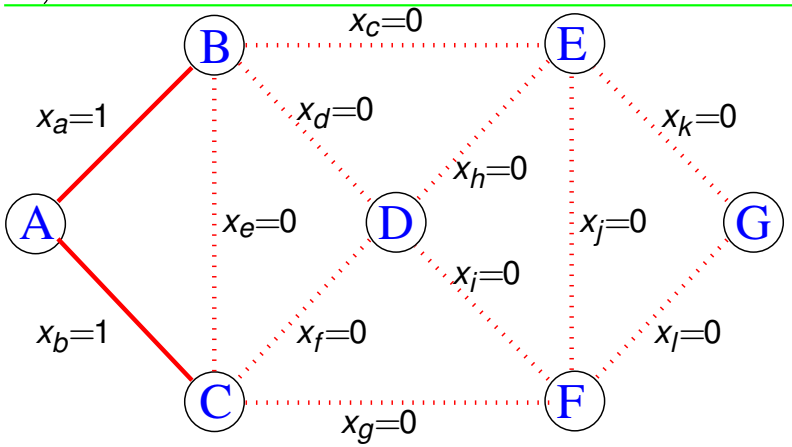
Para não sobrecarregar, em cada passo seguinte apresentaremos apenas os valores obtidos da solução ótima fracionária sem colocar os custos das arestas e seus nomes.



$$\begin{aligned} \min \quad & c_a x_a + c_b x_b + \cdots + c_l x_l \\ \text{s.a.} \quad & 0 \leq x_a \leq 1, \ldots, 0 \leq x_l \leq 1 \end{aligned}$$

Solução ótima = 0

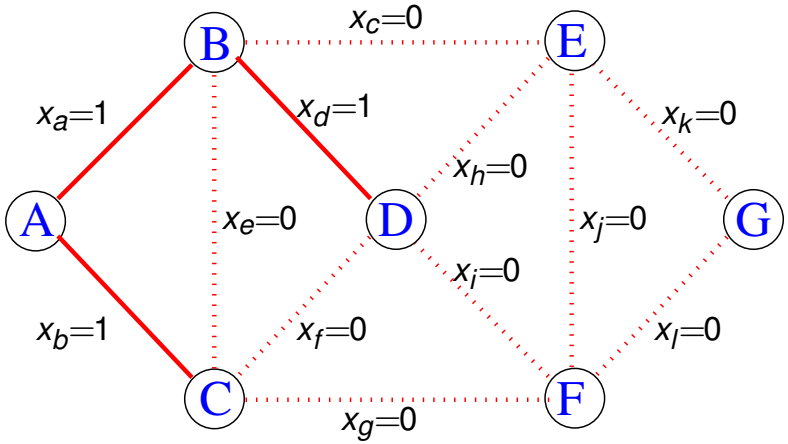
Apenas restrições $0 \leq x_e \leq 1$, $\forall e \in E$



$$\begin{aligned} \min \quad & c_a x_a + c_b x_b + \cdots + c_l x_l \\ \text{s.a.} \quad & x_a + x_b = 2 \\ & 0 \leq x_a \leq 1, \ldots, 0 \leq x_l \leq 1 \end{aligned}$$

Solução ótima = 12

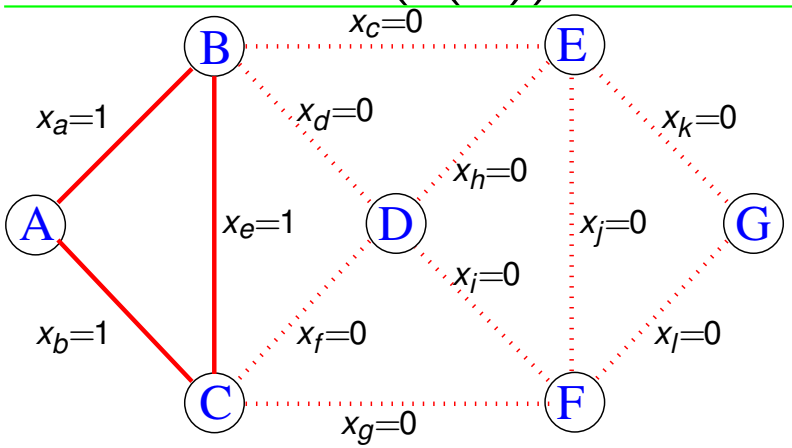
Adicionando $x(\delta(A)) = 2$



$$\min \quad c_a x_a + c_b x_b + \cdots + c_l x_l$$
$$\text{s.a.} \begin{cases} x_a + x_b = 2 \\ x_a + x_c + x_d + x_e = 2 \\ 0 \le x_a \le 1, \ldots, 0 \le x_l \le 1 \end{cases}$$

Solução ótima = 14

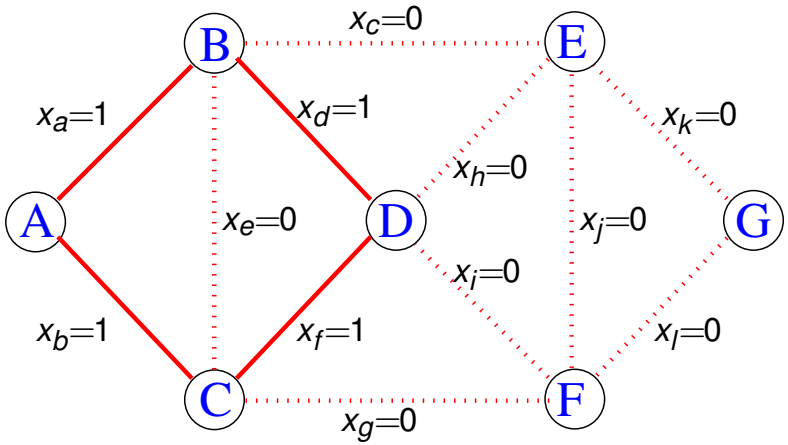
Adicionando $x(\delta(B)) = 2$



$$\min \quad c_a x_a + c_b x_b + \cdots + c_l x_l$$
$$\text{s.a.} \begin{cases} x_a + x_b = 2 \\ x_a + x_c + x_d + x_e = 2 \\ x_b + x_e + x_f + x_g = 2 \\ 0 \le x_a \le 1, \ldots, 0 \le x_l \le 1 \end{cases}$$

Solução ótima = 15

Adicionando $x(\delta(C)) = 2$

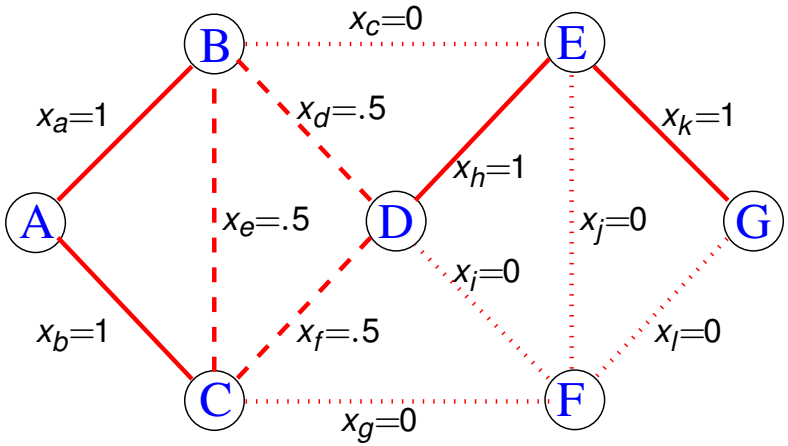


$$\min \quad c_a x_a + c_b x_b + \cdots + c_l x_l$$

$$\text{s.a.} \begin{cases} x_a + x_b = 2 \\ x_a + x_c + x_d + x_e = 2 \\ x_b + x_e + x_f + x_g = 2 \\ x_d + x_f + x_h + x_i = 2 \\ 0 \le x_a \le 1, \ldots, 0 \le x_l \le 1 \end{cases}$$

Adicionando $x(\delta(D)) = 2$

Solução ótima = 16

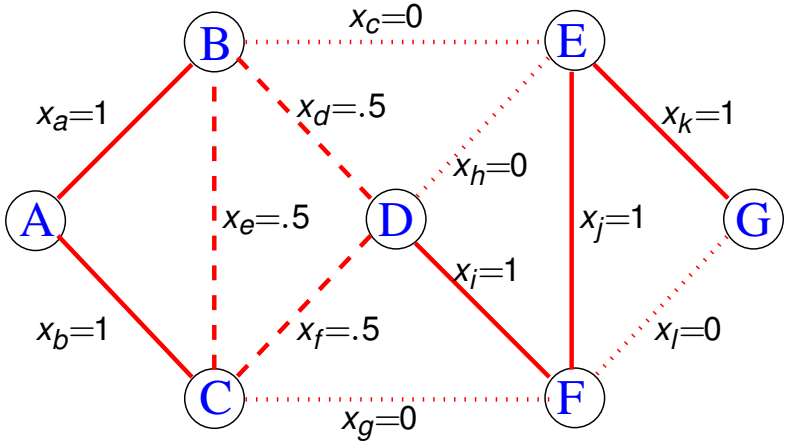


$$\min \quad c_a x_a + c_b x_b + \cdots + c_l x_l$$

$$\text{s.a.} \begin{cases} x_a + x_b = 2 \\ x_a + x_c + x_d + x_e = 2 \\ x_b + x_e + x_f + x_g = 2 \\ x_d + x_f + x_h + x_i = 2 \\ x_c + x_h + x_j + x_k = 2 \\ 0 \le x_a \le 1, \ldots, 0 \le x_l \le 1 \end{cases}$$

Adicionando $x(\delta(E)) = 2$

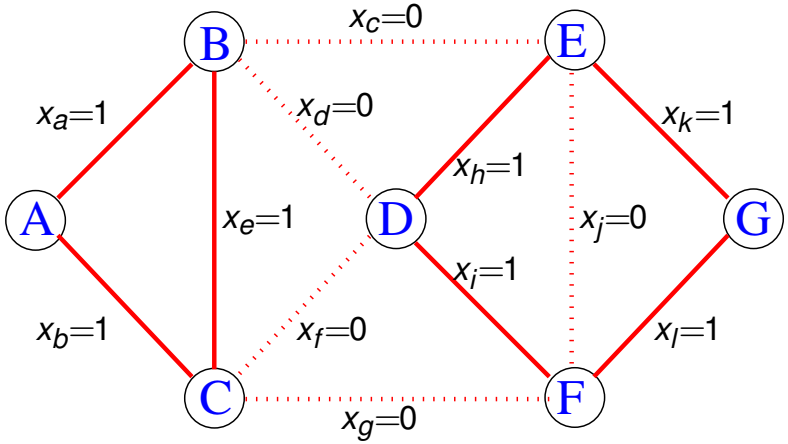
Solução ótima = 21,5



Adicionando $x(\delta(F)) = 2$

$$\begin{aligned}
\min \quad & c_a x_a + c_b x_b + \cdots + c_l x_l \\
\text{s.a.} \quad & \begin{cases}
x_a + x_b = 2 \\
x_a + x_c + x_d + x_e = 2 \\
x_b + x_e + x_f + x_g = 2 \\
x_d + x_f + x_h + x_i = 2 \\
x_c + x_h + x_j + x_k = 2 \\
x_g + x_i + x_j + x_l = 2 \\
0 \le x_a \le 1, \ldots, 0 \le x_l \le 1
\end{cases}
\end{aligned}$$

Solução ótima = 28.5

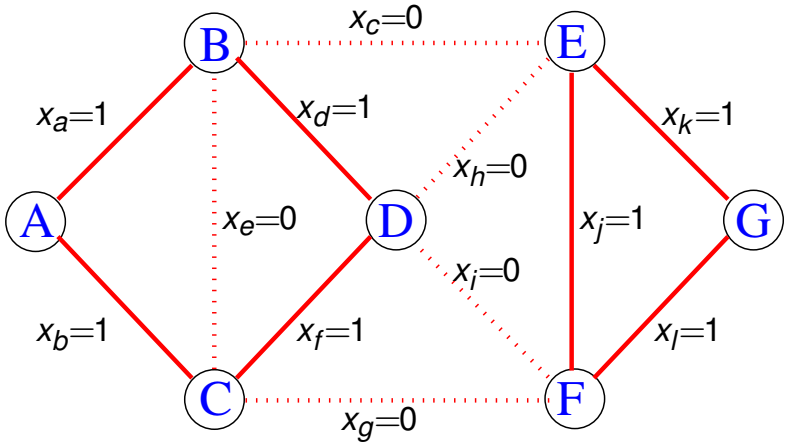


Adicionando $x(\delta(G)) = 2$

$$\begin{aligned}
\min \quad & c_a x_a + c_b x_b + \cdots + c_l x_l \\
\text{s.a.} \quad & \begin{cases}
x_a + x_b = 2 \\
x_a + x_c + x_d + x_e = 2 \\
x_b + x_e + x_f + x_g = 2 \\
x_d + x_f + x_h + x_i = 2 \\
x_c + x_h + x_j + x_k = 2 \\
x_g + x_i + x_j + x_l = 2 \\
x_k + x_l = 2 \\
0 \le x_a \le 1, \ldots, 0 \le x_l \le 1
\end{cases}
\end{aligned}$$

Solução ótima = 30

Todas as restrições $x(\delta(v)) = 2, \quad \forall v \in V$ foram inseridas.
Em geral, restrições estruturais que ocorrem pouco são inseridas todas de uma vez.
Agora vamos inserir as desigualdades de corte:



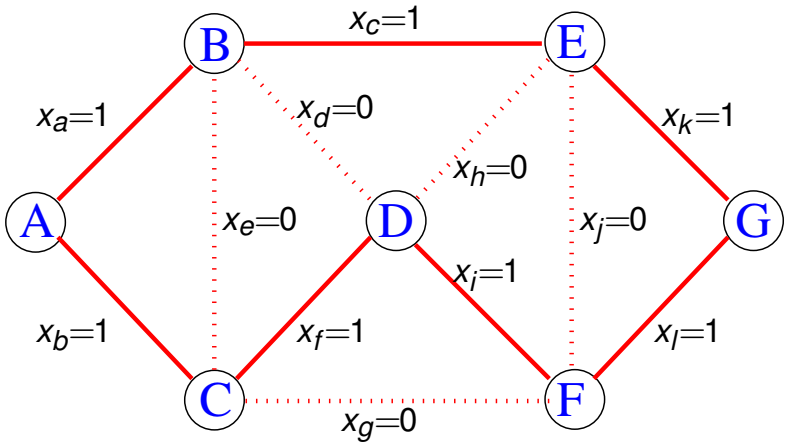
Adicionando desigualdade do corte mínimo que separa A e D (corte de valor 0)

$$x(\delta(S)) \geq 2$$

onde $S = \{A, B, C\}$

$$\begin{aligned} \min \quad & c_a x_a + c_b x_b + \cdots + c_l x_l \\ \text{s.a.} \quad & \left\{ \begin{array}{rcl} x_a + x_b & = & 2 \\ x_a + x_c + x_d + x_e & = & 2 \\ x_b + x_e + x_f + x_g & = & 2 \\ x_d + x_f + x_h + x_i & = & 2 \\ x_c + x_h + x_j + x_k & = & 2 \\ x_g + x_i + x_j + x_l & = & 2 \\ x_k + x_l & = & 2 \\ x_c + x_d + x_f + x_g & \geq & 2 \end{array} \right. \\ & 0 \leq x_a \leq 1, \ldots, 0 \leq x_l \leq 1 \end{aligned}$$

Solução ótima = 32



Adicionando desigualdade do corte mínimo que separa A e E (corte de valor 0)

$$x(\delta(S)) \geq 2$$

onde $S = \{A, B, C, D\}$

$$\min \quad c_a x_a + c_b x_b + \cdots + c_l x_l$$

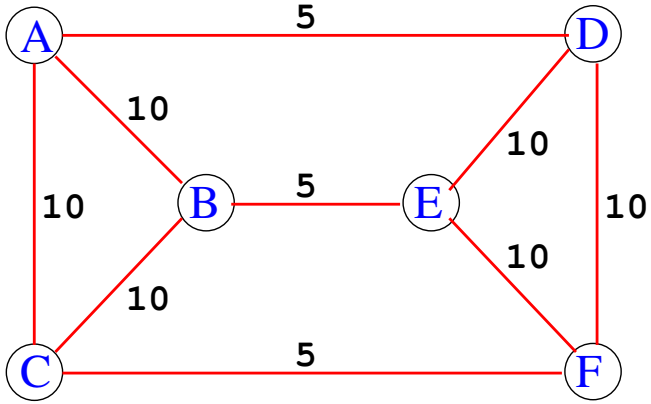
$$\text{s.a.} \begin{cases} x_a + x_b = 2 \\ x_a + x_c + x_d + x_e = 2 \\ x_b + x_e + x_f + x_g = 2 \\ x_d + x_f + x_h + x_i = 2 \\ x_c + x_h + x_j + x_k = 2 \\ x_g + x_i + x_j + x_l = 2 \\ x_k + x_l = 2 \\ x_c + x_d + x_f + x_g \geq 2 \\ x_c + x_g + x_h + x_i \geq 2 \\ 0 \leq x_a \leq 1, \ldots, 0 \leq x_l \leq 1 \end{cases}$$

Solução ótima = 33

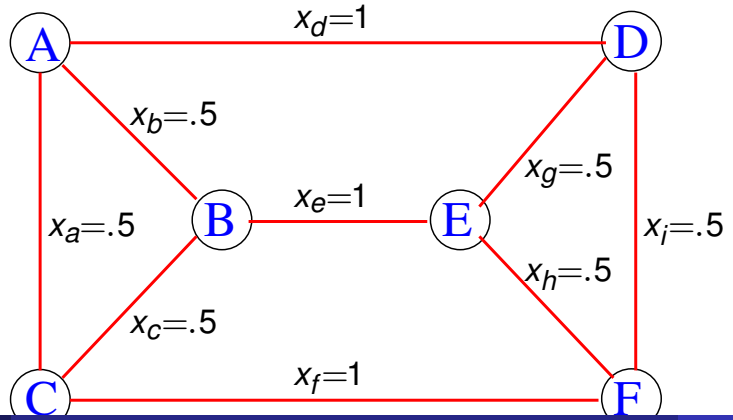
Chegamos em uma solução inteira!!! Solução ótima para TSP de valor 33.

Exemplo de solução ótima fracionária do programa relaxado do TSP

Grafo G (grafo envelope) e custos nas arestas:



Solução ótima fracionária de peso 45 (LP com restrições de grau e de co



Branch & Cut

Um algoritmo Branch & Cut é uma

- combinação do método Branch & Bound e
- geração de planos de corte durante a geração da árvore de B & B.

Parece simples, mas um algoritmo Branch & Cut envolve muitas sofistica

Estratégia:

⋆ **investir em cada nó para limitar crescimento da árvore e**

⋆ **delimitar problema**

- por estratégias de separação,
- uso de heurísticas primais em cada nó,
- métodos de ramificação,
- pré-processamento em cada nó
- gerenciamento de desigualdades válidas

Estratégias consideradas na separação

- Uso de heurísticas para encontrar planos de corte
  as vezes vale mais a pena usar uma heurística para obter planos de corte em tempo rápido que algoritmos que garantidamente determinam a existência de classe de planos de corte, mas a um alto custo computacional.
- Redução do número de desigualdades no nó
  Em sistemas grandes, remover as desigualdades que não estão justas
  Uma desigualdade $a \cdot x \leq \alpha$ é justa se a solução ótima do LP x^* satisfaz a desigualdade com igualdade.
  Isto diminui o número de restrições e permite que a resolução do sistema fique mais rápida.
- *Tailing off*
  Desigualdade de separação de alguns pontos pode gerar melhorias pequenas e sobrecarregar muito o sistema.

- Seleção de desigualdades
  - Escolha desigualdades de classes diferentes
    A escolha de desigualdades de diferentes classes é mais efetiva que concentrar em desigualdades de mesma classe.
  - Selecione as desigualdades mais violadas
    Para cada desigualdade válida encontrada $a \cdot x \leq \alpha$, calcule $a \cdot x' - \alpha$, onde x' é a solução da relaxação atual. Quanto maior for a diferença, maior a chance de crescimento do limitante inferior.
  - Selecione desigualdades que cobrem todo espaço de variáveis
    Um sistema linear "se adapta" a cada nova desigualdade, mudando o mínimo. Quando introduzimos desigualdades com grande quantidade de variáveis não nulas, a chance disso ocorrer é menor.

- Manutenção de um *Pool* (depósito) de desigualdades
  - Inserção no Pool de novo plano de corte
    Sempre que um novo plano de corte for encontrado, inserir no pool.
  - Ativação das desigualdades do Pool
    Uma desigualdade inserida no pool, pode ser um plano de corte para um nó. Assim, percorremos as desigualdades do pool, inserindo aquelas que são planos de corte para o nó corrente.
  - Eliminação das desigualdades do Pool
    Convém manter uma idade para cada desigualdade do pool. Cada vez que o pool é percorrido, aumenta se a idade das desigualdades que não foram ativadas. Posteriormente, eliminamos as desigualdades velhas. Permite que o pool não cresca exageradamente.
  - Manutenção de um pool de desigualdades por nó

Estratégias de delimitação

- Heurísticas Primais
  Aproveite informações da solução do programa relaxado em cada nó para obter soluções viáveis.
- Pre-processamento dos nós
  Fixação de variáveis por implicações lógicas. Considere o problema do TSP resolvido através de B & B com LP.

  Removendo as arestas fixadas em 0 e contraindo arestas fixadas em 1, podemos obter um grafo tal que:

  - se houver vértice de grau dois, podemos incluir as arestas incidentes na solução do nó.

  - se houver uma aresta cuja remoção desconecta o grafo, podemos podar o ramo deste nó.

Estratégias usadas na ramificação

- Ramificação por variáveis
  Escolha da variável com parte fracionária mais próxima de 0,5.
  Escolha da variável com parte fracionária mais distante de 0,5.
- Ramificação por restrição
  Ex.: Digamos que encontramos uma desigualdade que vale $x(\delta(S)) = 2{,}5$
  1. coloque a restrição $x(\delta(S)) = 2$ em um ramo e
  2. coloque a restrição $x(\delta(S)) \geq 3$ em outro ramo.

BRANCH-CUT-SIMPLIFICADO (P^o) % programa relaxado P^o

```
 1   LB ← −∞ % Lower bound
 2   x* ← Heurística sobre P^o (∅ se não encontrar solução viável)
 3   UB ← val(x*) (onde val(∅) = +∞)
 4   Ativos ← {P^o}
 5   enquanto Ativos ≠ ∅ faça
 6       escolha P ∈ Ativos e remova P de Ativos
 8       x ← solução ótima de P (x ← ∅ se P é inviável)
 9       se –Tailing On– então
10           enquanto x ≠ ∅ e consegue separar x faça
10               insira planos de corte separando x
11               x ← solução ótima de P (x ← ∅ se P é inviável)
12       se val(x) > LB então atualiza LB (se necessário)
13       se val(x) < UB % se val(x) ≥ UB , ramo é podado
14           se x é inteiro então
15               UB ← val(x);
15               x* ← x
17           senão
18               crie dois subproblemas P' e P'' a partir de P
19               Ativos ← Ativos ∪ {P' , P''}
20   devolva x
```

- Uma apresentação sobre o método branch & cut aplicado ao problema da árvore de Steiner (caso particular do Problema da Floresta de Steiner) pode ser encontrada em Ferreira & Wakabayashi'96, Capítulo 4.
  `http://www.ime.usp.br/~yw/livros/livro-new.ps.gz`