Programação Inteira

Flávio Keidi Miyazawa

IC - UNICAMP

Tópicos

- **Complexidade Computacional**
- **Programação Linear**
- **Programação Linear Inteira**
  - Modelagem de Problemas
  - Método Branch & Bound
  - Método por Planos de Cortes
  - Método Branch & Cut

Problemas de Otimização

Def.: *Um problema de otimização consiste em:*
Dado conjunto S, e função $f : S \to \mathbb{R}$, encontrar $x \in S$ tal que $f(x)$ é mínimo.

I.e.,

Encontrar x que
minimize $f(x)$
sujeito a $x \in S$

Nosso principal interesse será para o caso onde S é um domínio finito e enumerável.

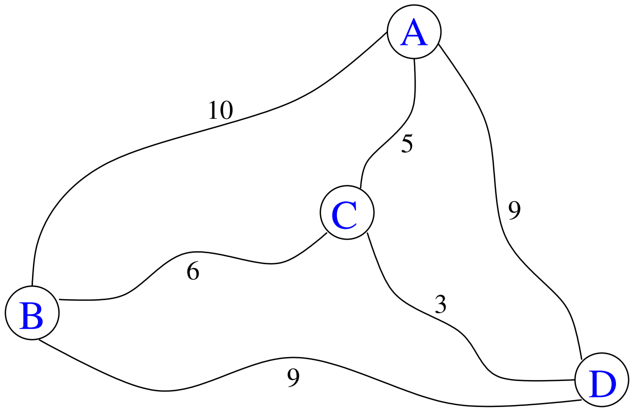
Problemas de Otimização Combinatória

- Domínio Finito (enumerável)
- Função de Otimização: Custos, Comprimentos, Quantidades
- Objetivo: Minimização ou Maximização

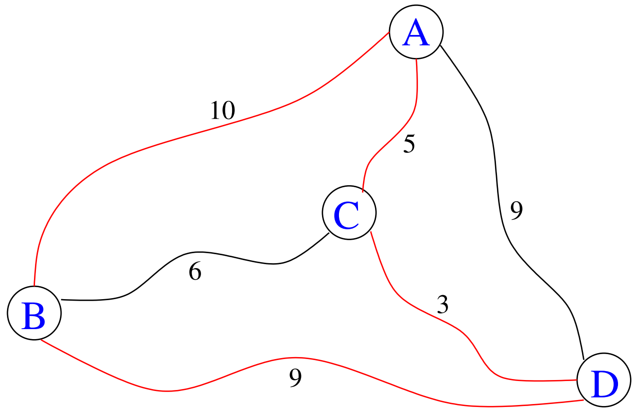
Exemplo: Problema do Caixeiro Viajante (TSP)

- *Entrada:*
  - Grafo não orientado: $G = (V, E)$,
  - Custos nas arestas: $c_e \geq 0, \forall e \in E$.
- *Objetivo:*
  Encontrar um *tour* (circuito hamiltoniano) de custo mínimo que visita cada vértice exatamente uma vez.

Encontre o circuito hamiltoniano de custo mínimo



Circuito hamiltoniano de custo mínimo: 27



Algoritmo Ingênuo para o TSP: Tentar todos os tours.

Complexidade: $O(n!)$, onde $n = |V|$.

Algoritmo *Bom* = Algoritmo Polinomial *(Cobham'64&Edmonds'65)*

Provavelmente não existam algoritmos rápidos para o TSP

Outros exemplos difíceis:

- Atribuição de Freqüências em Telefonia Celular
- Empacotamento de Objetos em Contêineres
- Escalonamento de Funcionários em Turnos de Trabalho
- Escalonamento de Tarefas em Computadores
- Classificação de Objetos
- Coloração de Mapas
- Projetos de Redes de Computadores
- Vários outros...

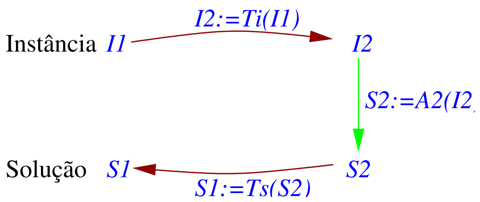
Complexidade Computacional

- Problemas de Decisão
  Simplificação dos problemas e sua formalização
  - Exemplo: Dado grafo $G = (V, E)$, existe um circuito hamiltoniano em G ?
  - Exemplo: Dado grafo completo $G = (V, E)$, $c : E \to \mathbb{Z}^+$ e um inteiro positivo K, existe um circuito hamiltoniano em G, de peso no máximo K ?
  - Exemplo: Dado um mapa M e um inteiro K, posso colorir M com K cores sem conflitos ?
  - Exemplo: Dado grafo completo $G = (V, E)$, $c : E \to \mathbb{Z}^+$, vértices s e t e um inteiro positivo K, existe um caminho em G, de s para t, de peso no máximo K ?

Redução polinomial entre problemas

P_1 é polinomialmente redutível a P_2 ($P_1 \leftarrow P_2$ ou $P_1 \preceq P_2$) se

- $\exists\, Ti$ que transforma instância I_1 de P_1 para instância I_2 de P_2
- $\exists\, Ts$ que transforma solução S_2 de I_2 para solução S_1 de I_1
- Ti e Ts têm complexidade de tempo polinomial



Conseqüências:

Se P_2 é "polinomial" então P_1 é "polinomial".

Se P_1 é "exponencial" então P_2 é pelo menos "exponencial".

Redução polinomial entre problemas de decisão

P_1 é polinomialmente redutível a P_2 ($P_1 \leftarrow P_2$ ou $P_1 \preceq P_2$) se

- $\exists\, T$ que transforma instância I_1 de P_1 para instância I_2 de P_2
- I_1 tem solução se e somente se I_2 tem solução
- T é polinomial

Conseqüências:

Se P_2 é "polinomial" então P_1 é "polinomial".

Se P_1 é "exponencial" então P_2 é pelo menos "exponencial".

Exemplo de reduções entre problemas

- Redução do problema de encontrar o máximo de um conjunto para o problema de ordenação do conjunto.
  Basta ordenar o conjunto, com os maiores primeiro e retornar o primeiro elemento
- Redução do problema de colorir um grafo com K cores para o problema de colorir com $K+1$ cores.
  Dado um grafo G (para o problema das K-cores), construímos um grafo G' inserindo um novo vértice em G e todas as arestas ligando o vértice novo com os antigos. G é K colorível se e somente se G' é $K+1$ colorível
- Redução do problema de ordenação de um conjunto para o problema de encontrar um caminho hamiltoniano em grafo orientado.
  Dado um conjunto C (a ordenar), construir um grafo orientado G cujo conjunto de vértices é C e existe aresta $u \to v$ se $u \leq v$. Um caminho hamiltoniano em G nos dá uma ordenação de C.

Classes de Complexidade:

P, NP, NP-Completo, NP-Difícil

$X \in$ P**:**

X pode ser decidido em tempo polinomial.

$X \in$ NP**:**

X tem certificado curto e verificável em tempo polinomial.

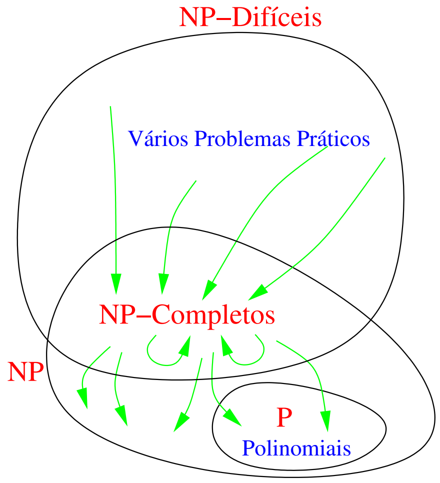
$X \in$ NP-Completo**:**

$X \in$ NP e $\forall Y \in NP, Y \preceq X$.

$X \in$ NP-Difícil**:**

$\forall Y \in$ NP, $Y \preceq X$, onde X não necessariamente pertence a NP.

Uma provável configuração das classes



Conjectura (Edmonds'65): P$\neq$NP ?

Prêmio de 1 milhão de US\$.

Problemas em **P**

- Dado um conjunto de números S, existe $k \in S$ tal que $k = \sum_{i \in S \setminus \{k\}} i$?
- Dado um grafo $G = (V, E)$, G é conexo ?
- Dado grafo completo $G = (V, E)$, $c : E \to \mathbb{Z}^+$, vértices s e t e um inteiro positivo K, existe um caminho em G, de s para t, de peso no máximo K ?
- Posso colorir um mapa com 4 cores sem conflitos ?

O primeiro problema NP-Completo

Def.: *Uma fórmula booleana está na forma normal conjuntiva (FNC) se é uma conjunção (conectivos $\wedge$) de cláusulas, cada cláusula contendo apenas conectivos $\vee$ entre literais.*

Exemplo: $f(x, y, z) = (\overline{x} \vee y \vee z) \wedge (\overline{y} \vee x) \wedge (x \vee z)$

Problema SAT: Dado uma fórmula booleana F na forma FNC, decidir se F tem uma atribuição para suas variáveis que a torne verdadeira.

Exemplo:
$f(x, y, z) = (\overline{x} \vee y \vee z) \wedge (\overline{y} \vee x) \wedge (x \vee z)$ é satisfeita para $x = 0$, $y = 0$ e $z = 1$.

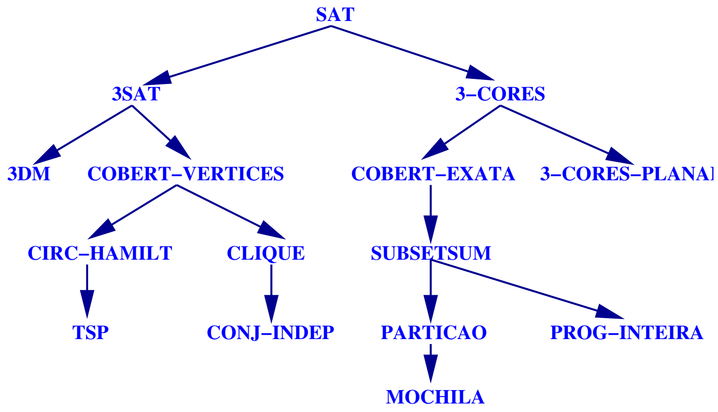
Verificação: Podemos verificar em tempo polinomial se uma atribuição satisfaz uma fórmula: SAT $\in$ NP

Teorema: *(Cook'71) O problema SAT é NP-completo.*

Problemas em **NP-Completo**

- Partição: Dado um conjunto de números S, existe $N \subseteq S$ tal que $\sum_{i \in N} i = \sum_{i \in S \setminus N} i$?
- Circuito Hamiltoniano: Dado grafo $G = (V, E)$, existe um circuito hamiltoniano em G ?
- TSP: Dado grafo completo $G = (V, E)$, $c : E \to \mathbb{Z}^*$ e um inteiro positivo K, existe um circuito hamiltoniano em G, de peso no máximo K ?
- Caminho Mínimo com pesos quaisquer Dado grafo completo $G = (V, E)$, $c : E \to \mathbb{Z}$, vértices s e t e um inteiro positivo K, existe um caminho em G, de s para t, de peso no máximo K ?
- 3-Cores: Posso colorir um mapa com 3 cores sem conflitos ?

Algumas reduções de problemas NP-completos



As primeiras reduções foram apresentadas por Karp [Kar72].

Problemas NP-difíceis

- Vários problemas práticos são NP-difíceis
  Exemplos:
  - Problema do Caixeiro Viajante
  - Atribuição de Freqüências em Telefonia Celular
  - Empacotamento de Objetos em Contêineres
  - Escalonamento de Funcionários em Turnos de Trabalho
  - Escalonamento de Tarefas em Computadores
  - Classificação de Objetos
  - Coloração de Mapas
  - Projetos de Redes de Computadores
  - Vários outros...
- **P$\neq$NP $\Rightarrow$ não existem algoritmos eficiêntes para problemas NP difíceis**
- Para uma lista grande de problemas NP-difíceis, veja Crescenzi e Kann [CK00].

Comparando tempos polinomiais e exponenciais

$f(n)$	$n=10$	$n=20$	$n=30$	$n=40$	$n=50$	$n=60$
n	.00001seg	.00002seg	.00003seg	.00004seg	.00005seg	.00006seg
n^2	.0001seg	.0004seg	.0009seg	.0016seg	.0025seg	.0036seg
n^3	.001seg	.008seg	.027seg	.064seg	.125seg	.216seg
n^5	.1seg	3.2seg	24.3seg	1.7min	5.2min	13.0min
2^n	.001seg	1.0seg	17.9min	12.7dias	35.7anos	366sec.
3^n	.059seg	58min	6.5anos	3855sec	2×10^8sec	1.3×10^{13}sec

Quando uma instrução é efetuada em 0.000001 segundos.

Comparando tempos polinomiais e exponenciais

$f(n)$	Computador atual	100×mais rápido	1000×mais rápido
n	N_1	$100N_1$	$1000N_1$
n^2	N_2	$10N_2$	$31.6N_2$
n^3	N_3	$4.64N_3$	$10N_3$
n^5	N_4	$2.5N_4$	$3.98N_4$
2^n	N_5	$N_5+6.64$	$N_5+9.97$
3^n	N_6	$N_6+4.19$	$N_6+6.29$

Fixando o tempo de execução

Para entender mais sobre a teoria de NP-completude, veja [Pap94, GJ79, CLR90].

Programação Linear

- **Definição**
- **Modelagem**
  – Problema da Dieta
  –– Portfolio problem
  –– Problema de transporte e fluxo
  –– Problema de emparelhamento em grafos bipartidos (atribuição)
- **Método de plano de cortes**
  –– Problema de emparelhamento

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

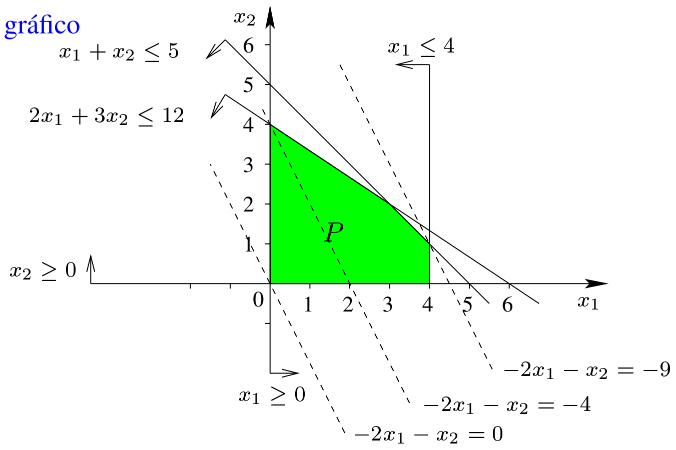
Programação Linear

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

$$
\begin{array}{ll}
\text{minimize} & c_1x_1 + c_2x_2 + \cdots + c_mx_m \\
\text{sujeito a} & \left\{ \begin{array}{rcl}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_n & \leq & b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_n & \geq & b_2 \\
\vdots & \vdots & \vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_n & = & b_n \\
x_i \in \mathbb{Q} & &
\end{array} \right.
\end{array}
$$

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

Exemplo gráfico



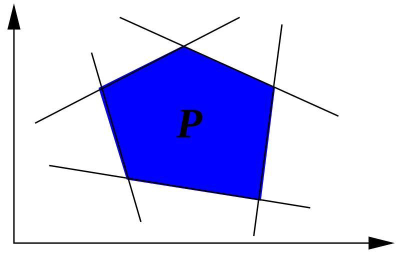
$$
\begin{array}{ll}
\text{minimize} & -2x_1 - x_2 \\
\text{sujeito a} & \left\{ \begin{array}{llcl}
x_1 & +x_2 & \leq & 5 \\
2x_1 & +3x_2 & \leq & 12 \\
x_1 & & \leq & 4 \\
x_1 & & \geq & 0 \\
 & x_2 & \geq & 0
\end{array} \right.
\end{array}
$$

Solução ótima: $x_1 = 4$ e $x_2 = 1$
Valor da solução ótima = -9

Def.: Chamamos o conjunto de pontos P,

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

como sendo um poliedro.



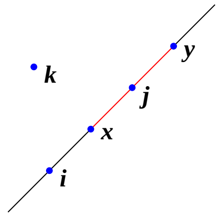
Def.: Se $y_i \in P$ e $\alpha_i \in \mathbb{Q}$, $i = 1, \ldots, n$ dizemos que $y = \alpha_1 y_1 + \alpha_2 y_2 + \cdots + \alpha_n y_n$ é uma combinação convexa dos pontos $y_i's$ se $\alpha_i \geq 0$ e $\alpha_1 + \cdots + \alpha_n = 1$

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

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

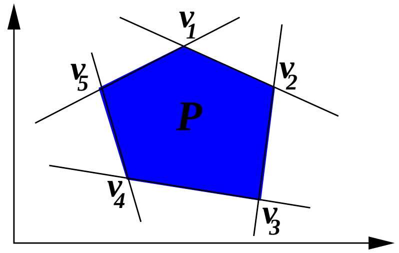
substituindo $\alpha_2 = 1 - \alpha_1$, temos

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



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

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

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

Algoritmos polinomiais para resolver PL:

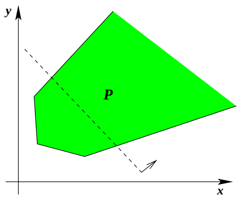
- Algoritmo dos elipsóides (Kachian'79) e
- Método dos pontos interiores (Karmarkar'84).

Algoritmos exponenciais para resolver PL:

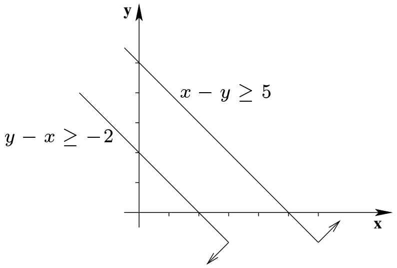
- Método simplex (Dantzig'47) (tempo médio polinomial).

Nem sempre encontramos uma solução ótima

- Sistema ilimitado



- Sistema inviável



Resolvedores de Sistemas Lineares

Programas comerciais

- CPLEX / ILOG: `www.ilog.com/products/cplex/`
- XPRESS: `www.dashoptimization.com/`
- OSL / IBM: `www.research.ibm.com/osl`

Programas livres

- GLPK / Gnu: `www.gnu.org/software/glpk/glpk.html`
- SoPlex / Zib: `www.zib.de/Optimization/Software/Soplex`

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)	4.78	110.00	29.00
ali_2 = Carne de Coelho (kg)	6.30	200.00	13.00
ali_2 = feijão cozido (kg)	1.48	170.00	15.00
ali_4 = Mamao Papaya (kg)	0.80	200.00	4.00
ali_3 = leite desnatado (lt)	0.85	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.,

$$\text{minimize } 4{,}78x_1 + 1{,}48x_2 + 0{,}85x_3$$

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

$$\begin{array}{ll} \text{minimize} & 4{,}78x_1 + 1{,}48x_2 + 0{,}85x_3 \\ \text{sujeito a} & \left\{ \begin{array}{rrrcl} 110x_1 + & 170x_2 + & 1150x_3 & \geq & 800 \\ 110x_1 + & 170x_2 + & 1150x_3 & \leq & 1200 \\ 29x_1 + & 15x_2 & & \geq & 10 \\ 29x_1 + & 15x_2 & & \leq & 18 \\ x_1 \geq 0 & x_2 \geq 0 & x_3 \geq 0 & & \end{array} \right. \end{array}$$

Resolvendo-se:

Obtemos uma dieta de 1,49 reais com

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

$x_2 = 0.67$ kg. de feijão cozido e

$x_3 = 0.60$ lt. de leite desnatado.

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

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)	4.78	1800.00	0.00
ali_2 = feijão cozido (kg)	1.48	490.00	10.00

Variáveis

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

Formulação Linear

$$\begin{array}{ll} \text{minimize} & 4{,}78x_1 + 1{,}48x_2 \\ \text{sujeito a} & \left\{ \begin{array}{rcrcl} 1800x_1 & + & 490x_2 & \geq & 800 \\ 1800x_1 & + & 490x_2 & \leq & 1200 \\ 0x_1 & + & 10x_2 & \geq & 60 \\ 0x_1 & + & 10x_2 & \leq & 90 \\ x_1 \geq 0 & & x_2 \geq 0 & & \end{array} \right. \end{array}$$

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 = Petrobrasa (petróleo/estatal)	9.0%
emp_2 = Vale do Rio Salgado (siderurgia)	10.2%
emp_3 = Votoratchim (siderurgia)	6.5%
emp_4 = Techaco (petróleo)	9.5%
emp_5 = Sanada (água/estatal)	8.5%

A recomendação dos especialistas é a seguinte:

- Recomenda se investir pelo menos 25% e no máximo 55% em empresas estatais.
- Petrobrasa e Techaco são empresas do mesmo setor (petróleo). Recomenda-se que o investimento nas duas não passe de 55 %.
- Vale do Rio Salgado e Votoratchim são do mesmo setor (siderurgia) recomenda que o investimento nas duas não passe de 45 %.
- Apesar da Vale do Rio Salgado 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 *Petrobrasa*
- x_2 quantidade de investimento na *Vale do Rio Salgado*
- x_3 quantidade de investimento na *Votoratchim*
- x_4 quantidade de investimento na *Techaco*
- x_5 quantidade de investimento na *Sanada*

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.

$$x_1 + x_5 \geq 25000$$
$$x_1 + x_5 \leq 55000$$

- Petrobrasa e Techaco 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 Salgado e Votoratchim 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 Salgado 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

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

Resolvendo-se:

Obtemos uma lucro estimado de 9094 reais investindo

$x_1 = 0$ na Petrobrasa, $x_2 = 12000$ na Vale do Rio Salgado,

$x_3 = 8000$ na Votoratchim, $x_4 = 55000$ na Techaco e

$x_5 = 25000$ na Sanada.

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.

Outras aplicações:

- 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.

Definições:

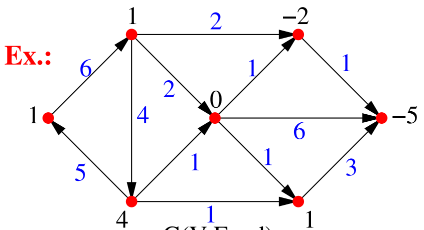
Seja $G = (V, E)$ um grafo orientado com função de capacidades nas arestas $c : E \to \mathbb{Q}^+$, demandas $b : V \to \mathbb{Q}$ e custos nas arestas $w : E \to \mathbb{Q}^+$.

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.*

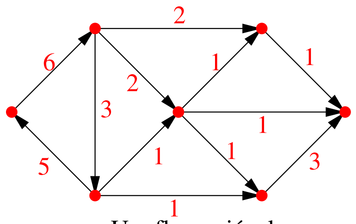
Def.: Dizemos que $x : E \to \mathbb{Q}^+$ é um fluxo em G se

$$\sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = b_i \qquad e \qquad 0 \le x_e \le c_e$$

Ex.:



G(V.E.c.d)



Um fluxo viável

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.

Formulação do Fluxo de custo mínimo

Encontrar x tal que

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

$$\sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = b_i$$

$$0 \le x_e \le c_e$$

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.*

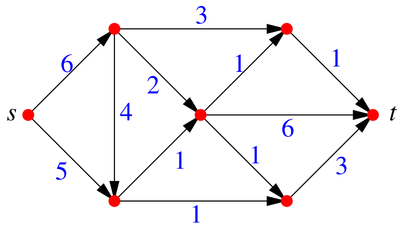
Subproblemas

- 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

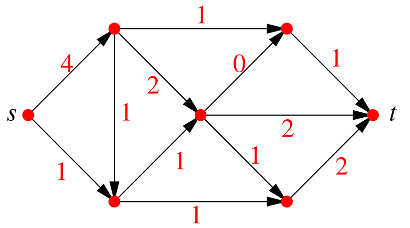
Fluxo Máximo de s para t (st-fluxo máximo)

- 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*

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

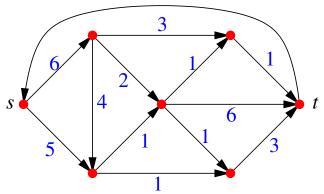
$$0 \le x_e \le c_e \quad \forall e \in E$$

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

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

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

st-Corte Mínimo

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

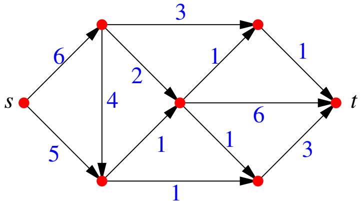
Def.: Definimos a capacidade de um st-corte S, como sendo a soma

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

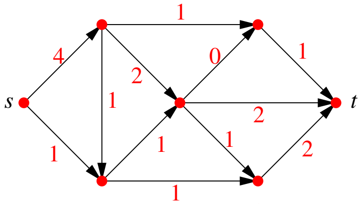
Problema st-CORTE MÍNIMO: Dado um grafo orientado $G = (V, E)$, capacidades $c : E \to \mathbb{Q}^+$ e vértices s e t, encontrar um st-corte de capacidade mínima.

Aplicações: Projeto de redes de conectividade, Classificação de dados (data mining), particionamento de circuitos VLSI, etc

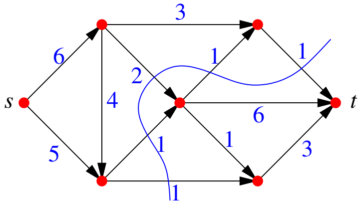
Teorema: *O valor de um st-fluxo máximo de s para t é igual à capacidade de um st-corte mínimo.*



G(V,E,c)



fluxo de valor 5



Corte de capacidade 5

Sabemos como encontrar um fluxo máximo de s para t, mas como encontrar um corte mínimo separando s e t ?

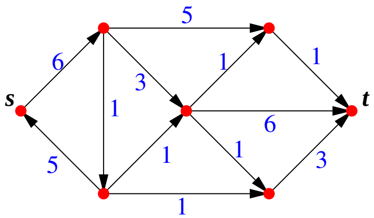
Exercícios:

- 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.

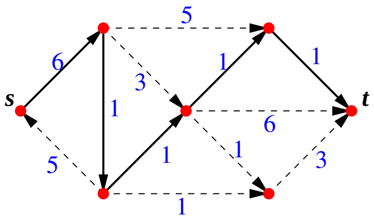
Um excelente livro sobre problemas em fluxo é dado em Ahuja et al [AMO93].

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.



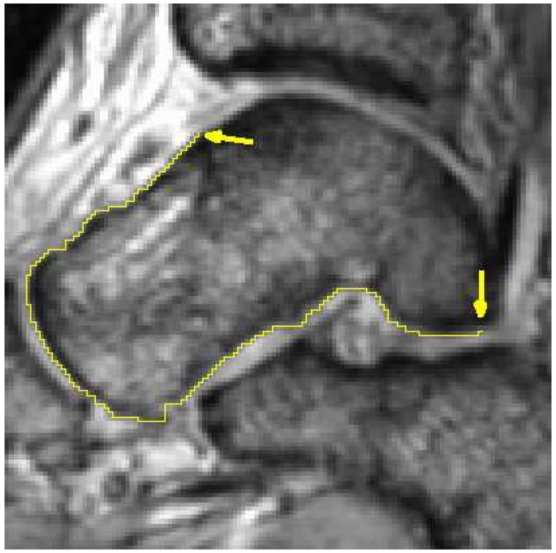
G(V,E,w)



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:

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

- 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., $x_e \in \{0,1\} \forall e \in E$.*

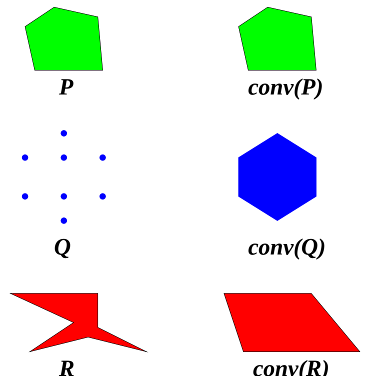
Teorema: *Se x é um ponto extremal ótimo de (P), então as arestas $e \in E$ onde $x_e = 1$ formam um caminho mínimo ligando s a t.*

Definições

Def.: *Dado um conjunto de pontos S, definimos o fecho convexo, denotado por $conv(S)$ o conjunto dos pontos formados pela 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 $conv(\{x,y\})$.

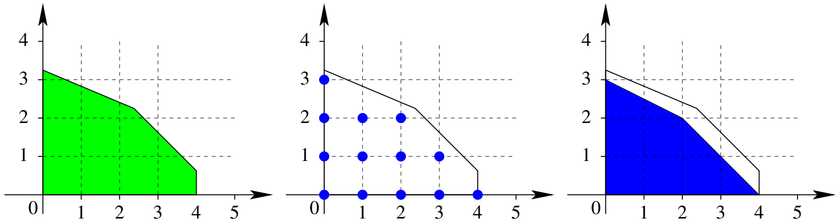
Exemplo:



Def.: *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.



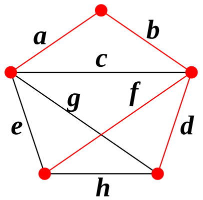
Poliedro P — Pontos inteiros I — $P_I := conv(I)$

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

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

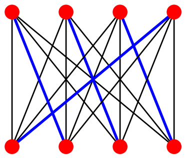
Exemplo: Seja $G = (V, E)$ o grafo abaixo com ordenação das arestas $E = (a, b, c, d, e, f, g, h)$ e T o subgrafo definido pelas arestas em vermelho.



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.

Formulação para Emparelhamento em Grafos Bipartidos

Formulação em Programação Inteira

$$
\begin{array}{ll}
\text{maximize} & \displaystyle\sum_{e \in E} c_e x_e \\
\text{sujeito a} & \left\{ \begin{array}{rcll} \displaystyle\sum_{e \in \delta(v)} x_e & \leq & 1 & \forall v \in V \\ x_e & \in & \{0,1\} & \forall e \in E \end{array} \right.
\end{array}
$$

A formulação nos dá pontos no espaço $\mathbb{R}^E$, cada ponto um emparelhamento.

O poliedro dos emparelhamentos bipartidos de um grafo $G = (V, E)$ é definido como

$$P_{Emp}(G) := conv\{\chi^M \in \mathbb{R}^E : M \text{ é um emparelhamento em } G\},$$

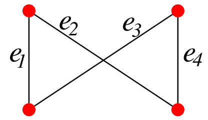
onde χ^M é o vetor de incidência do emparelhamento M.

Relaxação Linear

$$
(LP_{Emp}) \quad
\begin{array}{ll}
\text{maximize} & \displaystyle\sum_{e \in E} c_e x_e \\
\text{sujeito a} & \left\{ \begin{array}{rcll} \displaystyle\sum_{e \in \delta(v)} x_e & \leq & 1 & \forall e \in E \\ 0 \leq x_e & \leq & 1 & \forall e \in E \end{array} \right.
\end{array}
$$

Teorema: $LP_{Emp} = P_{Emp}$.
I.e., os vértices do poliedro relaxado do emparelhamento bipartido são inteiros.

Exemplo: Considere o seguinte grafo bipartido com custos unitários:



Programa Linear

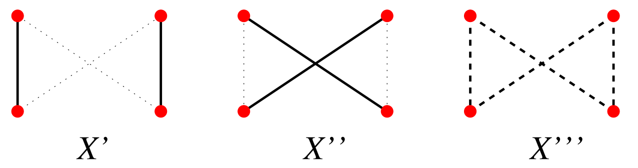
$$
\begin{array}{ll}
\text{maximize} & x_{e_1} + x_{e_2} + x_{e_3} + x_{e_4} \\
\text{sujeito a} & \left\{ \begin{array}{l} x_{e_1} + x_{e_2} \leq 1, \\ x_{e_3} + x_{e_4} \leq 1, \\ x_{e_1} + x_{e_3} \leq 1, \\ x_{e_2} + x_{e_4} \leq 1, \\ 0 \leq x_{e_1} \leq 1, \\ 0 \leq x_{e_2} \leq 1, \\ 0 \leq x_{e_3} \leq 1, \\ 0 \leq x_{e_4} \leq 1 \end{array} \right.
\end{array}
$$

Temos 4 variáveis binárias no 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' 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}.

Programação Linear Inteira

Os problemas resolvidos anteriormente por programação linear

- puderam ser resolvidos de maneira eficiênte (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

$$\begin{array}{ll} \text{minimize} & c_1x_1 + c_2x_2 + \cdots + c_mx_m \\ \text{sujeito a} & \left\{ \begin{array}{lcl} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_n & \leq & b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_n & \geq & b_2 \\ \vdots & \vdots & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_n & = & b_n \\ x_i \in \mathbb{Z} & & \end{array} \right. \end{array}$$

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 de uma árvore de ramificação: 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.

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

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

Coberturas, Empacotamentos e Partições

Coberturas, Empacotamentos e Partições são condições que ocorrem freqüêntemente 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**
$$\sum_{C \in \mathcal{C}_e} x_C = 1 \quad \forall e \in E.$$

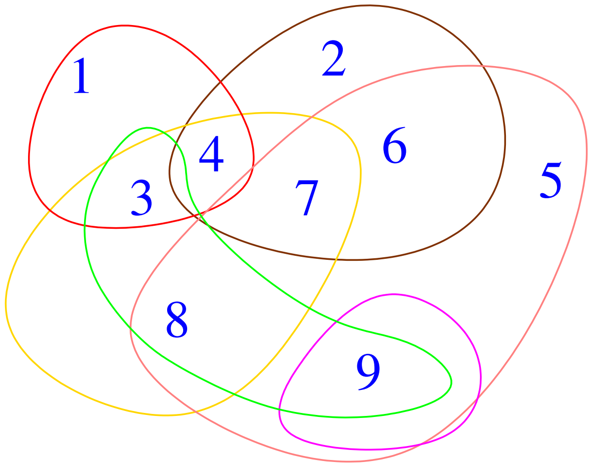
Problema da Cobertura por Conjuntos

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

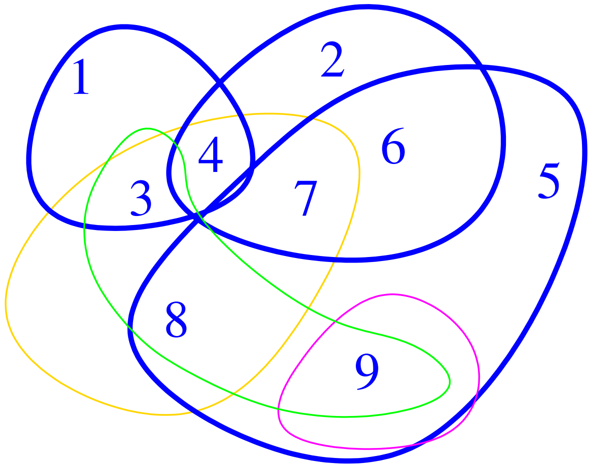
Problema COBERTURA POR CONJUNTOS: Dados conjunto E, subconjuntos $\mathcal{S}$ de E, custos $c(S)$, $S \in \mathcal{S}$, encontrar cobertura $\mathcal{S}' \subseteq \mathcal{S}$ que minimiza $c(\mathcal{S}')$.

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 de conjuntos:

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

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

Formulação do Problema da Cobertura de 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.

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

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.

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{i \in F} f_i y_i + \sum_{ij \in E} c_{ij} x_{ij} \\ \text{sujeito a} & \left\{ \begin{array}{rcll} \displaystyle\sum_{ij \in E} x_{ij} & = & 1 & \forall j \in C, \\ x_{ij} & \leq & y_i & \forall ij \in E, \\ y_i & \in & \{0,1\} & \forall i \in F \\ x_{ij} & \in & \{0,1\} & \forall i \in F \text{ e } j \in C. \end{array} \right. \end{array}$$

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 Árvore de Steiner

Seja G um grafo e T (terminais) um subconjunto de vértices de V_G.

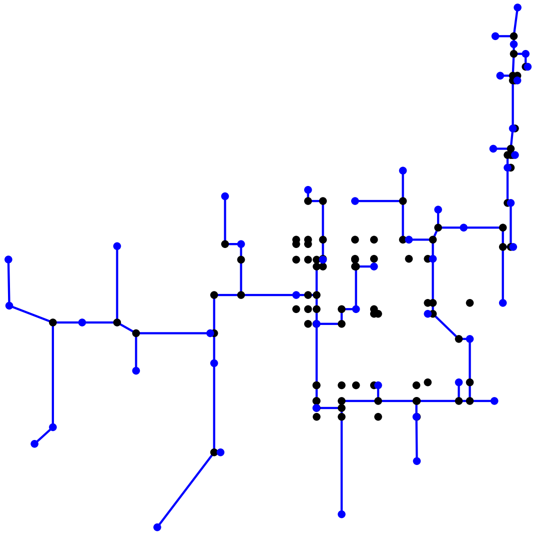
Def.: *Uma Árvore de Steiner de G é qualquer árvore H de G tal que todo elemento de T está em H.*

Problema Árvore de Steiner: Dados um grafo G, um custo c_e em $\mathbb{Q}_{\geq}$ para cada aresta e e um conjunto T de V_G, encontrar uma Árvore de Steiner H que minimize $c(H)$.

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:



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

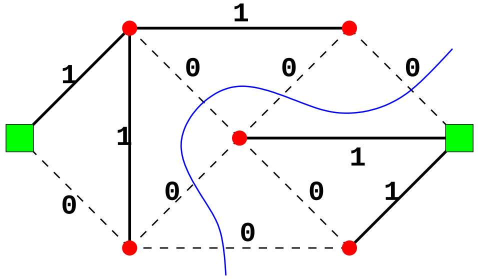
Note que se S é inválido

- 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
- 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:

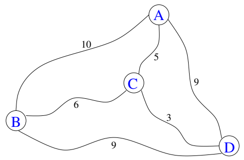
$$\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{e\in E} c_e x_e & \\
\text{sujeito a} & \left\{\begin{array}{llll}
\displaystyle\sum_{e\in\delta(S)} x_e & \geq & 1 & \forall S \subset V: \\
& & & S \text{ separa terminais.} \\
& & & \text{(desigualdades de corte)} \\
x_e & \in & \{0,1\} & \forall e \in E
\end{array}\right. &
\end{array}$$

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

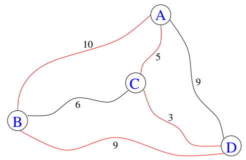
A primeira restrição impõe que todo corte separador deve ter pelo menos uma aresta que pertença à solução.

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/chtsp`
- CONCORDE: Applegate, Bixby, Chvátal, Cook'01: Branch & cut para TSP Soluções para instâncias de até 15112 vértices.

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

$$\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{e\in E} c_e x_e & \\
\text{sujeito a} & \left\{\begin{array}{llll}
\displaystyle\sum_{e\in\delta(v)} x_e & = & 2 & \forall v \in V \\
\displaystyle\sum_{e\in\delta(S)} x_e & \geq & 2 & \forall S \subset V, \quad S \neq \emptyset \\
& & & \text{(restrições de subcircuito)} \\
x_e & \in & \{0,1\} & \forall e \in E
\end{array}\right. &
\end{array}$$

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.

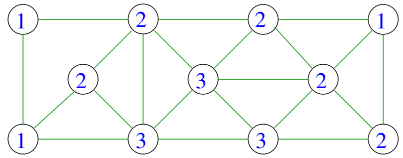
Survivable Network Design Problem

(Na falta de uma tradução boa)

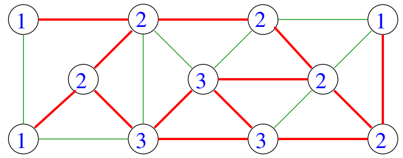
- Neste problema temos vários pontos em uma rede de telecomunicações e alguns pontos que devem ser conectados com requisitos de conectividade.
- Requisitos de conectividade são dados por uma função $f : V \times V \to \mathbb{N}$ que indica o grau de conectividade entre pares de nós.
- Dado dois nós 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:

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

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

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

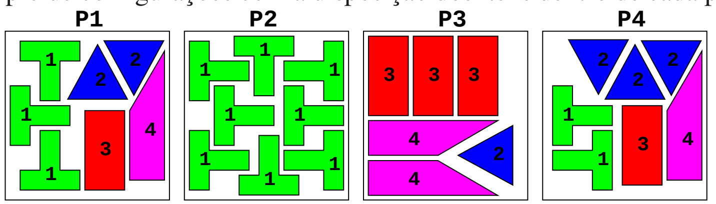
Teorema: *O problema da separação relativo às desigualdades de corte pode ser resolvido em tempo polinomial.*

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_i.
- 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:

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

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

Formulação:

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

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".

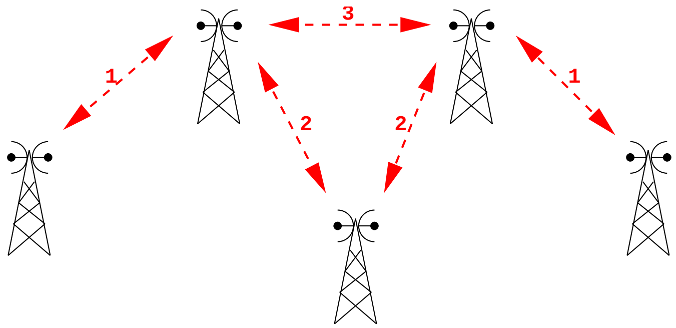
- **Alternativas no lado direito de igualdades**
  Se deve valer a igualdade $ax = \alpha$ onde $\alpha \in \{\alpha_1, \ldots, \alpha_k\}$, usamos novas variáveis binárias $\Delta_1, \ldots, \Delta_k$ onde $\Delta_i = 1$ se e somente se a igualdade satisfeita é $ax = \alpha_i$.

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

- **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 módulo:

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

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

onde M é um número suficientemente grande.

Se $\Delta_{ij} = 0$, a primeira restrição nos dá

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

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).

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 empacotamento.

Formulação em PLI:

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

Relaxação do PLI:

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

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.

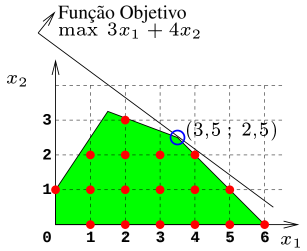
Branch & Bound e Programação Linear Inteira

- Modelar o problema como problema de programação linear inteira
- Relaxar a formulação para programação linear
- Enumerar o espaço de soluções, através de uma árvore: Cada nó da árvore representa um programa linear. Cada programa linear (nó) é ramificado enquanto houver valores fracionários em sua solução.
- Se um nó leva a atribuições inviáveis, ou a soluções não promissoras, podar o nó.

Considere o programa linear inteiro:

$$
\begin{array}{ll}
\text{minimize} & 3x_1 + 4x_2 \\
\text{sujeito a} & \left\{ \begin{array}{rrcl} -3x_1 & +2x_2 & \leq & 2 \\ x_1 & +3x_2 & \leq & 11 \\ x_1 & +x_2 & \leq & 6 \\ \multicolumn{4}{l}{x_1 \geq 0, \quad x_2 \geq 0} \\ \multicolumn{4}{l}{x_i \in \mathbb{Z}^*} \end{array} \right.
\end{array}
$$

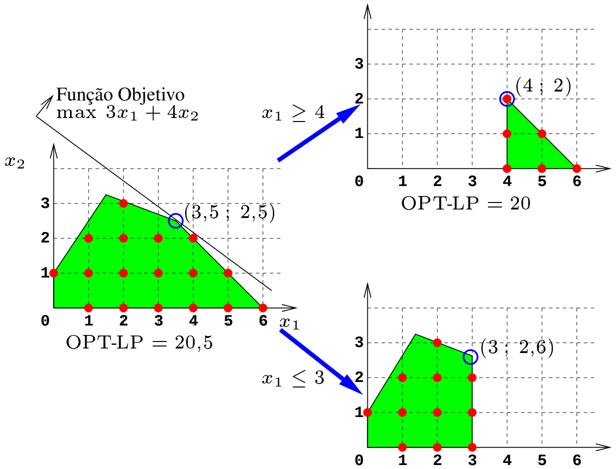
Representação gráfica do programa relaxado e os pontos inteiros:



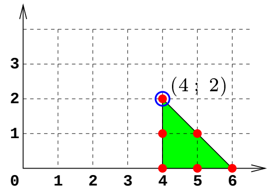
OPT-LP = 20,5

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:

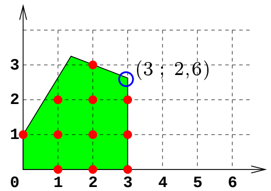


Um dos programas já tem solução inteira e não precisamos continuar sua ramificação:



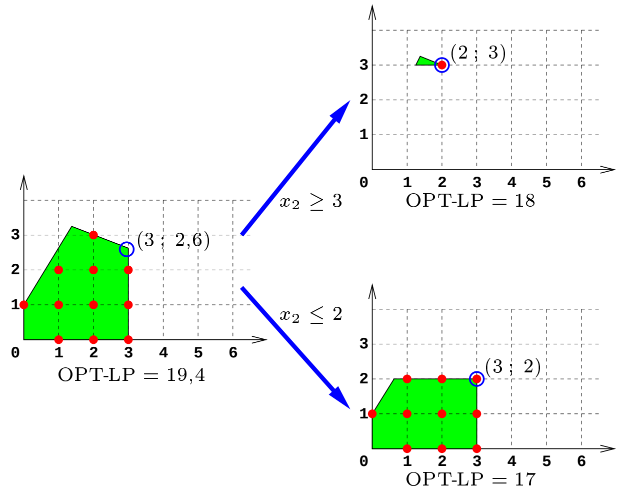
OPT-LP = 20

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



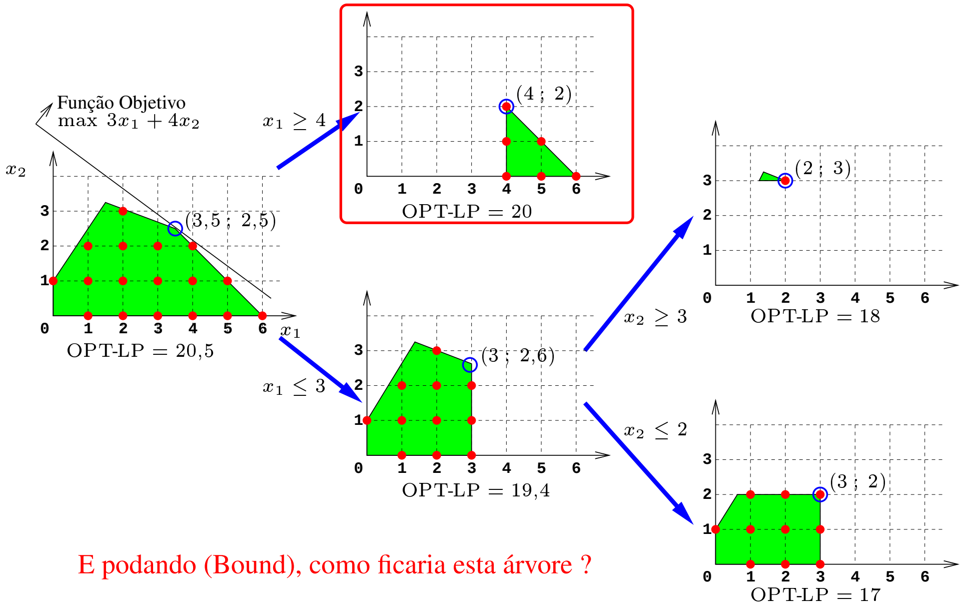
OPT-LP = 19,4

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



Desta vez, os dois novos programas tem soluções inteiras e podemos parar o processo.

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
- **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)

- **Representando a árvore de Branch & Bound:**
  Em geral não armazenamos 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.

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).

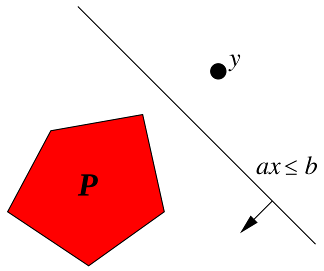
Estratégia:

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 $\times$ 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 $ay > b$.

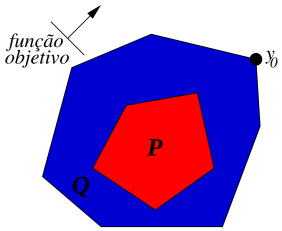


Teorema: *(Grötschel, Lovász, Schrijver'81) O problema de otimização de um programa linear pode ser resolvido em tempo polinomial se e somente se o problema da separação para o poliedro do programa linear pode ser resolvido em tempo polinomial.*

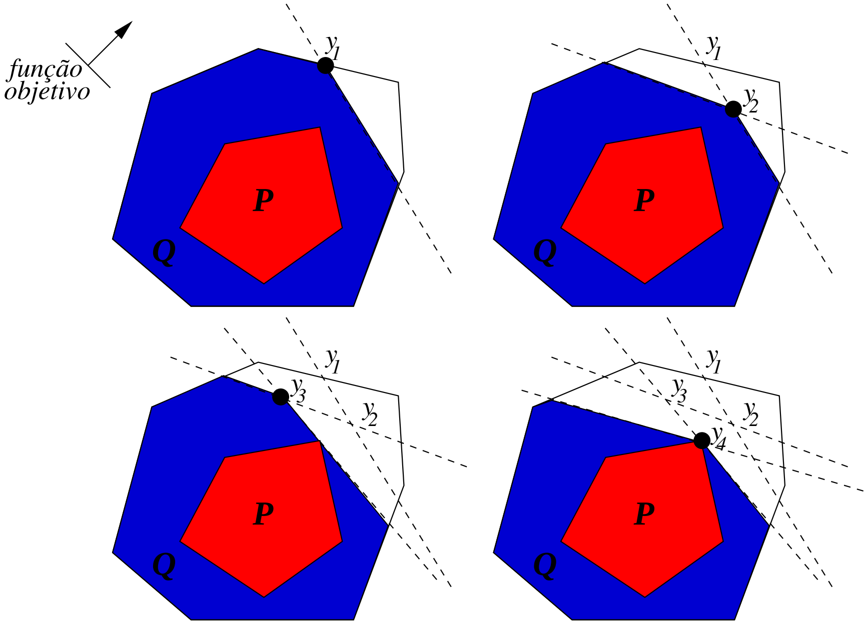
ALGORITMO PLANOS DE CORTE(P, c) Dantzig, Fulkerson e Johnson'54

P poliedro (não necessariamente explícito),
c função objetivo

```
Q ← {Poliedro inicial}
y ← OPT-LP(Q, c)
enquanto y pode ser separado de Q faça
    seja ax ≤ b desigualdade de P que separa y
    Q ← Q ∩ {x : ax ≤ b}
    y ← OPT-LP(Q, c)
se Q = ∅ retorne ∅
senão retorne y,
```



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



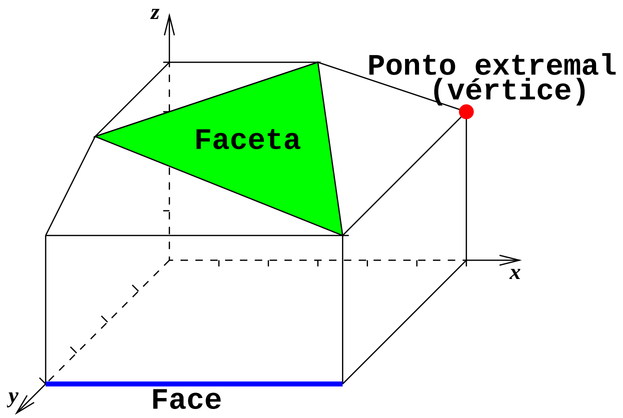
Que tipo de desigualdade são as melhores para serem inseridas ?

Desigualdades Válidas, Faces e Facetas

Seja P um poliedro.

- $ax \leq b$ é uma desigualdade válida para P se $P \subseteq \{x : ax \leq b\}$.
- Um conjunto F é dito ser uma face de P se existe desigualdade válida $ax \leq b$ tal que $F = P \cap \{x : ax \leq b\}$
- Uma face F de P é dita ser própria se $F \neq P$.
- Uma face F de P é dita ser não-trivial se $\emptyset \neq F \neq P$.
- Uma face não-trivial F é dita ser uma faceta de P se F não está contida em nenhuma outra face própria de P.

Exemplo de Faces e Facetas



Observe que as melhores desigualdades válidas são aquelas que induzem facetas.

Geração de Planos de Corte para programa linear inteiro

Estratégia:

1. Formular o problema como problema de programação linear inteira
2. Relaxar a formulação inteira para programação linear
3. Repetidamente inserir planos de corte

Como obter os planos de corte ?

- Gerando desigualdades válidas a partir do programa linear relaxado
- Gerando desigualdades pelas propriedades das soluções do problema

Gerando desigualdades a partir do programa linear

Cortes de Chvátal:

1. Suponha que queremos soluções inteiras, $x \in \mathbb{Z}^m$ para um poliedro $P := \{x : Ax \leq b\}$, $A \in \mathbb{Q}^{n\times m}$ e $b \in \mathbb{Q}^n$.
2. Idéia: Combinar as linhas do sistema $Ax \leq b$ para obter uma desigualdade $ax \leq t$ tal que o vetor a é inteiro e t não.
3. Inserir a nova desigualdade $ax \leq \lfloor t \rfloor$

Problema do Emparelhamento

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.
Queremos obter $P_{Emp}(G) := conv\{\chi^M \in \mathbb{R}^E : M \text{ é um emparelhamento}\}$.

Formulação Inteira do problema do Emparelhamento:

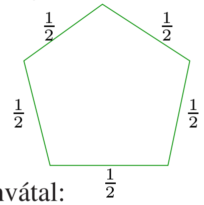
$$\begin{array}{ll} \text{maximize} & \sum_{e\in E} c_e x_e \\ \text{sujeito a} & \left\{\begin{array}{rcll} \sum_{e\in\delta(v)} x_e & \leq & 1 & \forall v \in V, \\ x_e & \in & \{0,1\} & \forall e \in E. \end{array}\right. \end{array}$$

Relaxação linear

$$\begin{array}{ll} & \text{maximize} \quad \sum_{e\in E} c_e x_e \\ (P) & \text{sujeito a} \quad \left\{\begin{array}{rcll} \sum_{e\in\delta(v)} x_e & \leq & 1 & \forall v \in V, \\ x_e & \geq & 0 & \forall e \in E. \end{array}\right. \end{array}$$

Sabemos que quando G é bipartido $P = P_{Emp}$.
E quando G não é bipartido, a igualdade ocorre ?
Não.

Se P tem custos unitários, a seguinte atribuição é ótima com valor 2,5



Vamos gerar um corte de Chvátal:

1. Seja $U \subseteq V$ tal que $|U| \geq 3$ é impar.
2. Some as desigualdades $\sum_{e\in\delta(u)} x_e \leq 1$ para todo $u \in U$.
3. Obtemos: $2\sum_{e\in E(U)} x_e + \sum_{e\in\delta(u)} x_e \leq |U|$, onde $E(U)$ são arestas com ambos extremos em U.
4. Ignorando o segundo termo: $2\sum_{e\in E(U)} x_e \leq |U|$
5. Dividindo por dois: $\sum_{e\in E(U)} x_e \leq \frac{|U|}{2}$
6. Lado esquerdo é inteiro e o direito é fracionário: $\sum_{e\in E(U)} x_e \leq \lfloor\frac{|U|}{2}\rfloor$
7. Novo corte (que separa atribuição acima): $\sum_{e\in E(U)} x_e \leq \frac{|U|-1}{2}$

Considere o novo programa linear (P') com os cortes de Chvátal:

$$\begin{array}{ll} \text{maximize} & \displaystyle\sum_{e\in E} c_e x_e \\ (P') \ \text{sujeito a} & \left\{\begin{array}{rcll} \displaystyle\sum_{e\in\delta(v)} x_e & \leq & 1 & \forall v \in V, \\ \displaystyle\sum_{e\in E[S]} x_e & \leq & \dfrac{|S|-1}{2} & \forall S \subseteq V,\ |S| \text{ ímpar}, \\ x_e & \geq & 0 & \forall e \in E \end{array}\right. \end{array}$$

onde $E[S] := \{e \in E : e \text{ tem ambos extremos em } S\}$.

Teorema: *(Edmonds'65) $P' = P_{Emp}$.*

Teorema: *(Padberg, Rao'82) O problema da separação do poliedro relaxado do emparelhamento pode ser resolvido em tempo polinomial.*
Veja código em
`http://elib.zib.de/pub/Packages/mathprog/index.html`

Corolário: *O problema do emparelhamento máximo em grafos quaisquer pode ser resolvido em tempo polinomial.*

Outros métodos para geração de planos de corte para programas lineares inteiros

- Cortes de Gomory (Gomory'58)
- Cortes de Schrijver (Schrijver'80)

Estes métodos de planos de corte (Chvátal, Gomory e Schrijver)

1. são independentes do problema
2. garantem uma solução ótima inteira em tempo finito
3. nem sempre são bem sucedidos como no problema de emparelhamento
4. em geral são lentos e apresentam pequena melhora em cada iteração

Como gerar planos de corte bons ?

Use as propriedades estruturais das soluções do seu problema em particular para conseguir desigualdades válidas boas, se possível que induzem facetas

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 ?

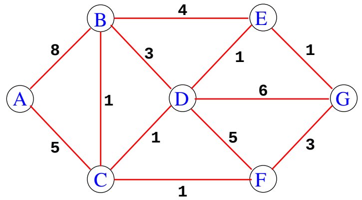
Como testar se dois vértices s e t estão separados por um corte $(S,\overline{S})$ que viola a desigualdade de corte, i.e., está ocorrendo

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

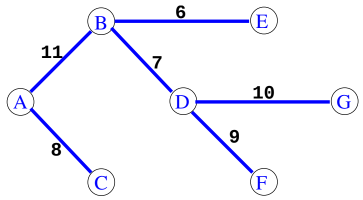
podemos executar o algoritmo para st-corte mínimo.
Note que o algoritmo deve ser executado não para o grafo original, mas onde cada aresta e tem capacidade x_e.
Se o valor do st-corte mínimo for menor que 2, então o corte gerado viola a desigualdade de corte e podemos inseri-la como desigualdade válida.

- 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:



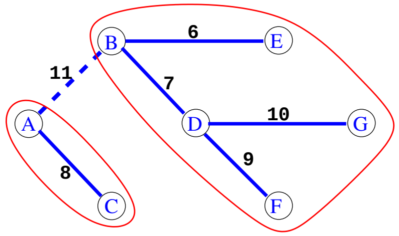
Grafo $G = (V, E)$

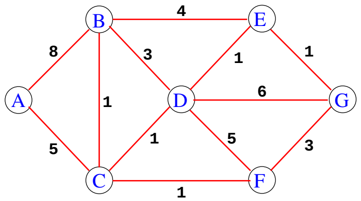


Árvore T de Cortes de Gomory-Hu de G

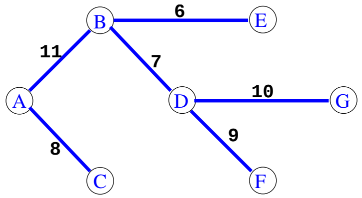
Cada aresta na árvore de cortes de Gomory-Hu representa um corte.
A capacidade do corte é o peso da aresta em T.

Exemplo do corte de capacidade 11 representado pela aresta (A, B) em T



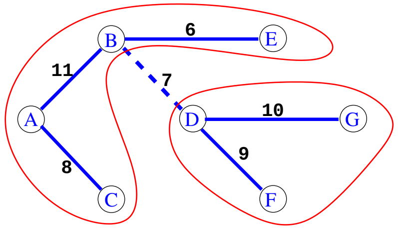


Grafo $G = (V, E)$



Árvore T de Cortes de Gomory-Hu de G

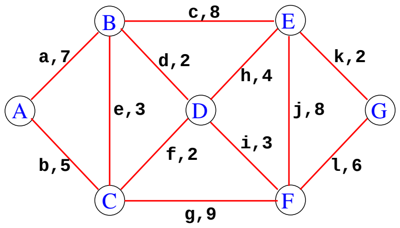
Corte mínimo que separa u e v em G é dado pelas componentes geradas pela remoção da aresta de capacidade mínima no caminho único entre u e v em T.



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

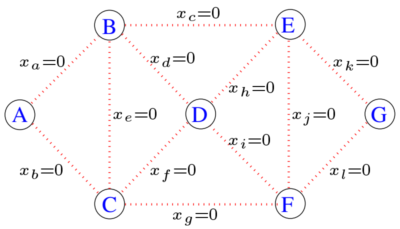
Separação para o TSP

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

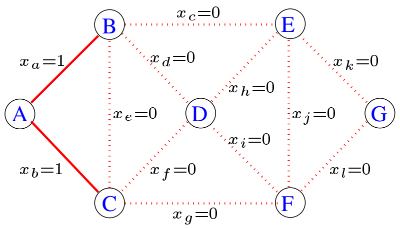
Para não sobrecarregar, em cada passo seguinte apresentaremos apenas os valores obtidos da solução ótima fracionária sem colocar os custos das arestas e seus nomes.



$$\begin{array}{ll} \min & c_a x_a + c_b x_b + \cdots + c_l x_l \\ \text{s.a.} & \left\{ \ 0 \le x_a \le 1, \ldots, 0 \le x_l \le 1 \right. \end{array}$$

Solução ótima = 0

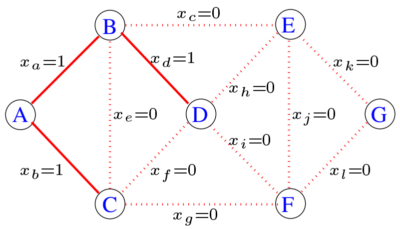
Apenas restrições $0 \le x_e \le 1, \ \forall e \in E$



$$\begin{array}{ll} \min & c_a x_a + c_b x_b + \cdots + c_l x_l \\ \text{s.a.} & \left\{ \begin{array}{rcl} x_a + x_b & = & 2 \\ 0 \le x_a \le 1, \ldots, 0 \le x_l \le 1 \end{array} \right. \end{array}$$

Solução ótima = 12

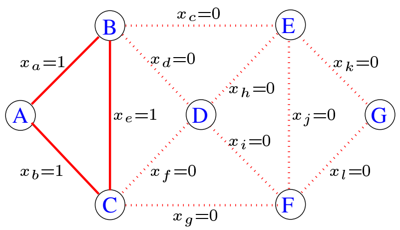
Adicionando $x(\delta(A)) = 2$



$$\begin{array}{ll} \min & c_a x_a + c_b x_b + \cdots + c_l x_l \\ \text{s.a.} & \left\{ \begin{array}{rcl} x_a + x_b & = & 2 \\ x_a + x_c + x_d + x_e & = & 2 \\ 0 \le x_a \le 1, \ldots, 0 \le x_l \le 1 \end{array} \right. \end{array}$$

Solução ótima = 14

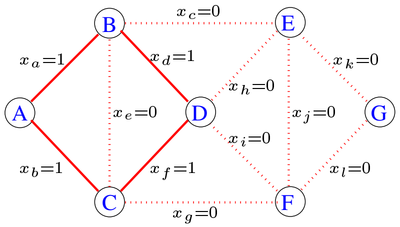
Adicionando $x(\delta(B)) = 2$



$$\begin{array}{ll} \min & c_a x_a + c_b x_b + \cdots + c_l x_l \\ \text{s.a.} & \left\{ \begin{array}{rcl} x_a + x_b & = & 2 \\ x_a + x_c + x_d + x_e & = & 2 \\ x_b + x_e + x_f + x_g & = & 2 \\ 0 \le x_a \le 1, \ldots, 0 \le x_l \le 1 \end{array} \right. \end{array}$$

Solução ótima = 15

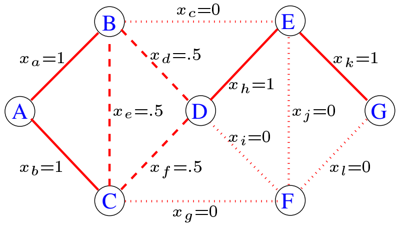
Adicionando $x(\delta(C)) = 2$



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

$$\begin{array}{ll} \min & c_a x_a + c_b x_b + \cdots + c_l x_l \\ \text{s.a.} & \left\{ \begin{array}{rcl} x_a + x_b & = & 2 \\ x_a + x_c + x_d + x_e & = & 2 \\ x_b + x_e + x_f + x_g & = & 2 \\ x_d + x_f + x_h + x_i & = & 2 \\ \multicolumn{3}{l}{0 \le x_a \le 1, \ldots, 0 \le x_l \le 1} \end{array} \right. \end{array}$$

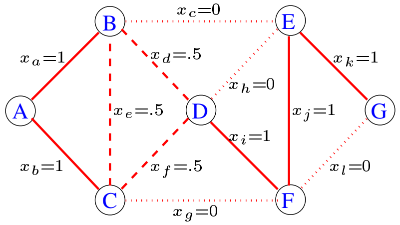
Solução ótima = 16



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

$$\begin{array}{ll} \min & c_a x_a + c_b x_b + \cdots + c_l x_l \\ \text{s.a.} & \left\{ \begin{array}{rcl} x_a + x_b & = & 2 \\ x_a + x_c + x_d + x_e & = & 2 \\ x_b + x_e + x_f + x_g & = & 2 \\ x_d + x_f + x_h + x_i & = & 2 \\ x_c + x_h + x_j + x_k & = & 2 \\ \multicolumn{3}{l}{0 \le x_a \le 1, \ldots, 0 \le x_l \le 1} \end{array} \right. \end{array}$$

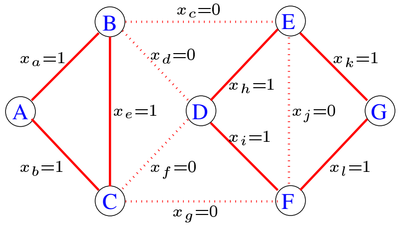
Solução ótima = 21,5



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

$$\begin{array}{ll} \min & c_a x_a + c_b x_b + \cdots + c_l x_l \\ \text{s.a.} & \left\{ \begin{array}{rcl} x_a + x_b & = & 2 \\ x_a + x_c + x_d + x_e & = & 2 \\ x_b + x_e + x_f + x_g & = & 2 \\ x_d + x_f + x_h + x_i & = & 2 \\ x_c + x_h + x_j + x_k & = & 2 \\ x_g + x_i + x_j + x_l & = & 2 \\ \multicolumn{3}{l}{0 \le x_a \le 1, \ldots, 0 \le x_l \le 1} \end{array} \right. \end{array}$$

Solução ótima = 28.5

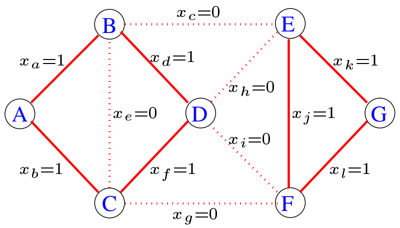


Adicionando $x(\delta(G)) = 2$

$$\begin{array}{ll} \min & c_a x_a + c_b x_b + \cdots + c_l x_l \\ \text{s.a.} & \left\{ \begin{array}{rcl} x_a + x_b & = & 2 \\ x_a + x_c + x_d + x_e & = & 2 \\ x_b + x_e + x_f + x_g & = & 2 \\ x_d + x_f + x_h + x_i & = & 2 \\ x_c + x_h + x_j + x_k & = & 2 \\ x_g + x_i + x_j + x_l & = & 2 \\ x_k + x_l & = & 2 \\ \multicolumn{3}{l}{0 \le x_a \le 1, \ldots, 0 \le x_l \le 1} \end{array} \right. \end{array}$$

Solução ótima = 30

Todas as restrições $x(\delta(v)) = 2$ para todo vértice v foram inseridas.
Agora vamos inserir as desigualdades de corte:



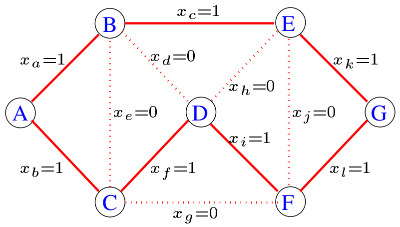
Adicionando desigualdade do corte mínimo que separa A e D (corte de valor 0)

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

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

$$\begin{array}{ll} \min & c_a x_a + c_b x_b + \cdots + c_l x_l \\ \text{s.a.} & \left\{ \begin{array}{rcl} x_a + x_b & = & 2 \\ x_a + x_c + x_d + x_e & = & 2 \\ x_b + x_e + x_f + x_g & = & 2 \\ x_d + x_f + x_h + x_i & = & 2 \\ x_c + x_h + x_j + x_k & = & 2 \\ x_g + x_i + x_j + x_l & = & 2 \\ x_k + x_l & = & 2 \\ x_c + x_d + x_f + x_g & \ge & 2 \\ \multicolumn{3}{l}{0 \le x_a \le 1, \ldots, 0 \le x_l \le 1} \end{array} \right. \end{array}$$

Solução ótima = 32



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

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

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

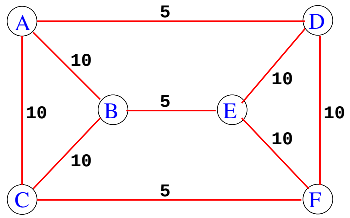
$$\begin{array}{ll} \min & c_a x_a + c_b x_b + \cdots + c_l x_l \\ \text{s.a.} & \left\{ \begin{array}{rcl} x_a + x_b & = & 2 \\ x_a + x_c + x_d + x_e & = & 2 \\ x_b + x_e + x_f + x_g & = & 2 \\ x_d + x_f + x_h + x_i & = & 2 \\ x_c + x_h + x_j + x_k & = & 2 \\ x_g + x_i + x_j + x_l & = & 2 \\ x_k + x_l & = & 2 \\ x_c + x_d + x_f + x_g & \ge & 2 \\ x_c + x_g + x_h + x_i & \ge & 2 \\ \multicolumn{3}{l}{0 \le x_a \le 1, \ldots, 0 \le x_l \le 1} \end{array} \right. \end{array}$$

Solução ótima = 33

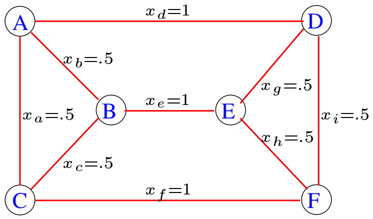
Chegamos em uma solução inteira!!! Solução ótima para TSP de valor 33.

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 corte):



Chvátal'75: apresenta separação de soluções como acima (comb inequalities)

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 sofisticações

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.
  Em vez de separar estes pontos, ir direto para a ramificação.

- 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.

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 $(P, c, \min)$

```
y* ← {Solução heurística, ∅ se não obteve solução}
U ← val(y*)
Ativos ← {P}
enquanto Ativos ≠ ∅ faça
    escolha Q ∈ Ativos e remova Q de Ativos
    y ← OPT-LP(Q, c, min)
    enquanto y ≠ ∅ e encontra plano ax ≤ γ que separa y faça
        Q ← Q ∩ {x : ax ≤ γ}
        y ← OPT-LP(Q, c, min)
    se val(y) < U então
        se y é inteiro então
            U ← val(y);
            y* ← y
        senão
            seja y_i uma variável fracionária em y e α seu valor
            Q' ← Q ∩ {x : x_i ≤ ⌊α⌋}
            Q'' ← Q ∩ {x : x_i ≥ ⌈α⌉}
            Ativos ← Ativos ∪ {Q', Q''}
devolva y*.
```


Algoritmos Gulosos

- **Método Construtivo:**
  – Adicionando items ou
  – Extendendo "soluções parciais"
- **Sem backtracking** – Não reconsidera itens já adicionados
- **Escolha Gulosa:**
  – Próximo item ou extenção é escolhido de maneira gulosa

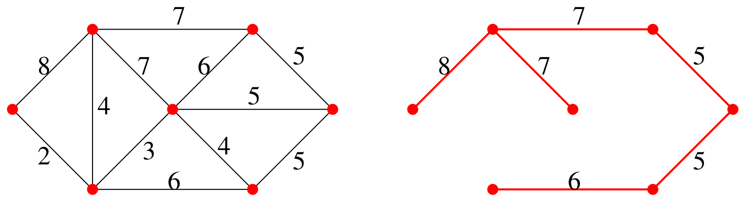
Algoritmos Gulosos Ótimos

- **Ingredientes para algoritmos gulosos ótimos**
  - Propriedade de Escolha Gulosa:
    Solução ótima global pode ser encontrada fazendo escolha (gulosa) ótima local.
  - Subestrutura ótima:
    Solução ótima contém (sub)soluções ótimas para subproblemas
- **Exemplos**
  - Árvore geradora de peso máximo
  - Árvore (de compressão) de Huffman

Problema da Árvore Geradora de Peso Máximo

Problema MST: Dado um grafo conexo não orientado $G = (V, E)$ e uma função de custo nas arestas $c : E \to \mathbb{R}$, encontrar uma árvore geradora de G de custo máximo.

Aplicações: Projeto de redes de computadores, determinação de rotas aéreas e como heurística para diversos outros problemas.



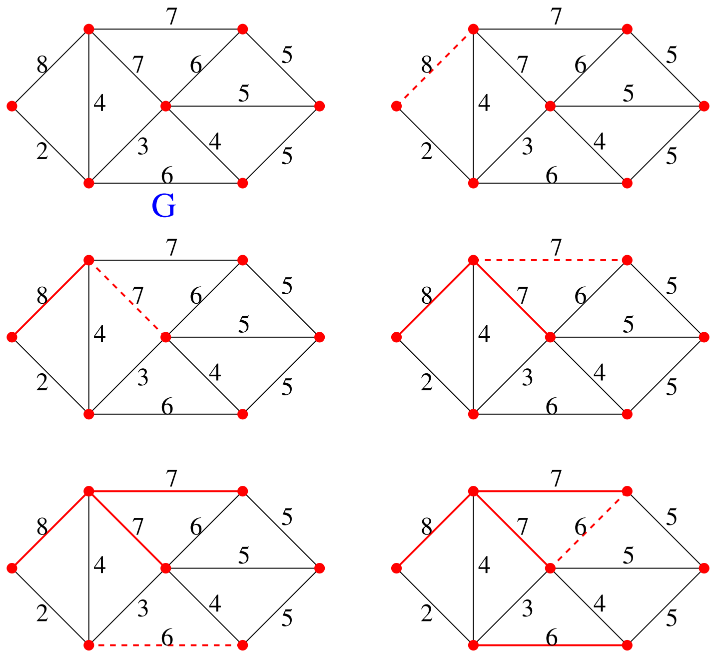
G Árvore Geradora de peso 38

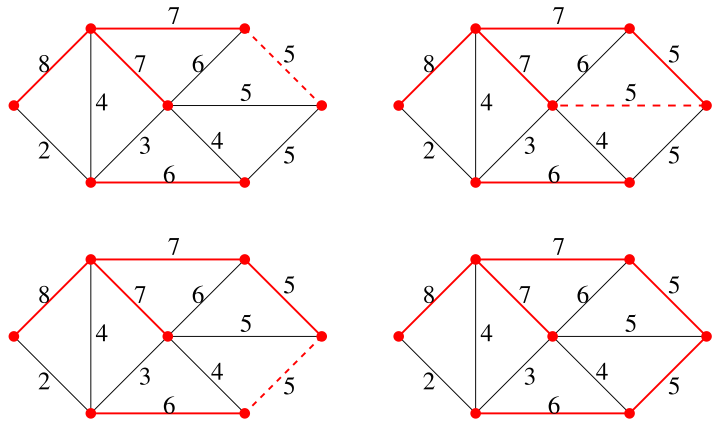
- **Algoritmo guloso: Selecionar as arestas de maior custo primeiro.**
- **Descartar arestas que formam circuito com as escolhidas anteriormente.**

KRUSKAL $(G = (V, E))$

$A \leftarrow \emptyset$ % Conjunto de arestas de T.
$(e_1, e_2, \ldots, e_m) \leftarrow$ (ordenação de E, tal que $c(e_i) \geq c(e_{i+1})$
para $i \leftarrow 1$ até m faça
    se e_i liga componentes distintas em $G[A]$ então
        $A \leftarrow A + e_i$
devolva $G[A]$

onde $G[A]$ é o grafo gerado pelas arestas em A.





Teorema: *O algoritmo de Kruskal encontra uma árvore geradora de peso máximo.*

Para ver uma teoria sobre algoritmos gulosos ótimos, procure sobre *matroids* em [CLR90].

Exercício: Faça uma implementação eficiente do algoritmo de Kruskal.

Código de Huffman

- **Codificação Comum de Textos:**
  - Alfabeto de tamanho fixo
  - Freqüencias distintas para cada elemento do alfabeto
- **Compressão pelo código de huffman**
  - Codificação dos elementos do alfabeto com tamanho variável de bits
  - Caracteres mais freqüêntes com códigos menores
  - Caracteres menos freqüêntes com códigos maiores
  - Codificação sem perda
- **Programas/formatos que usam o Código de Huffman ou alguma variante:**
  – pkZIP, lha, gz, zoo, arj, JPEG, MPEG

Exemplo: Alfabeto: $\mathcal{A} = \{a, b, c, d, e, f\}$.

	a	b	c	d	e	f
Freqüência de cada letra	45	13	12	16	9	5
Codificação usando 3 bits	000	001	010	011	100	101
Codificação de tamanho variável	0	101	100	111	1101	1100

Arquivo com 100.000 caracteres:

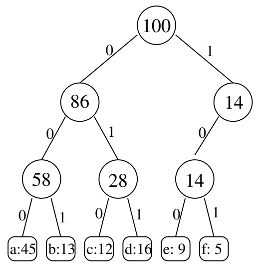
- Arquivo texto normal: 3 bits $\times$ 100.000 = 300.000 bits
- Codificação por Código de Huffman

$$\begin{aligned}
\sum_{s\in\mathcal{A}} freq(s) \;\times\; 100.000 \times |codigo(s)| & \\
&= (0.45\times 1 \text{ bit } + 0.13\times 3 \text{ bits } + 0.12\times 3 \text{ bits } + \\
&\quad 0.16\times 3 \text{ bits } + 0.9\times 4 \text{ bits } + 0.5\times 4 \text{ bits })\times 100.000 \\
&= 45.000 + 39.000 + 36.000 + 48.000 + 36.000 + 20.000 \\
&= 224.000 \text{ bits}
\end{aligned}$$

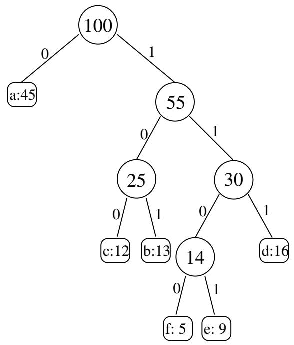
- 25% de economia: $\frac{300.000-224.000}{300.000} = 0.2533.$

Representação por árvore binária

- Código Prefixo
  - Nenhum símbolo tem codificação que é prefixo de outro
- Representação em árvore binária 0/1
  - Ramo esquerdo nos dá um bit 0
  - Ramo direito nos dá um bit 1
  - Símbolo: representado pelo trajeto da raiz até uma folha



Árvore para código fixo



Árvore para código variável

Representando a seqüência abc:

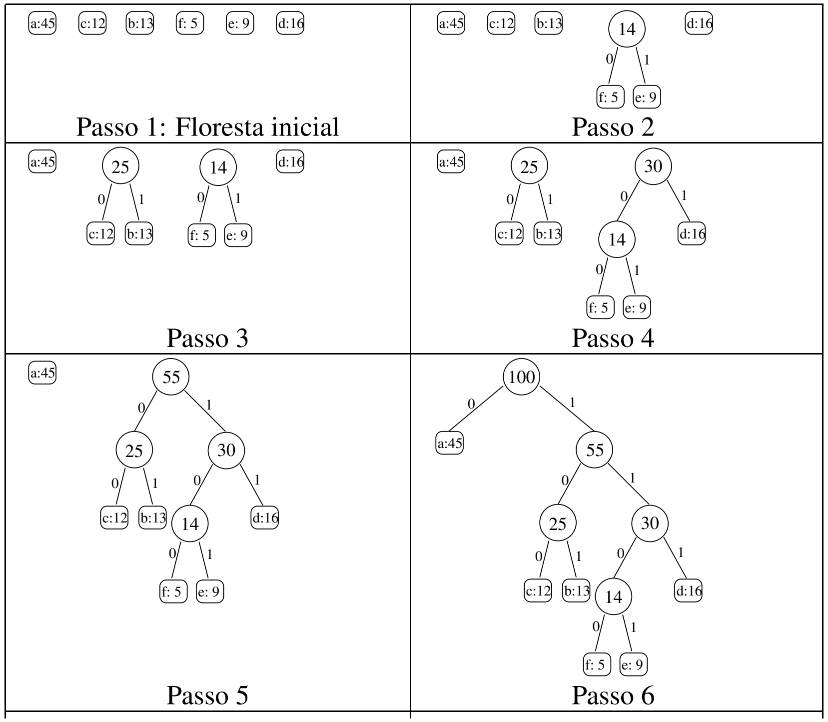
1. Na codificação fixa: $000||001||010 = 000001010$.
2. Na codificação variável: $0||101||100 = 0101100$.

Note que não precisamos representar separação entre a codificação dos símbolos

ALGORITMO HUFFMAN($\mathcal{A}$)

seja A uma floresta com cada símbolo como um nó isolado
enquanto A não é conexo faça
    $\forall$ componente c de A, $f(c) \leftarrow$ freqüência dos símbolos em c.
    seja a e b duas componentes com menor freqüência
    ligue a e b por um novo nó c e arestas (c, a) e (c, b)
    coloque label 0 para aresta (c, a) e label 1 para aresta (c, b)
retorne a árvore gerada

Teorema: *O algoritmo de Huffman produz uma árvore de código prefixo ótima. Veja demonstração em [CLR90].*



Escalonamento de Tarefas

Problema ESCALONAMENTO: Dados uma lista de tarefas $L = (J_1, \dots, J_n)$, cada uma com tempo $t(J_i)$ e m máquinas idênticas, encontrar uma partição de L, $(M_1, \dots, M_m)$, tal que $\max_i t(M_i)$ é mínimo.

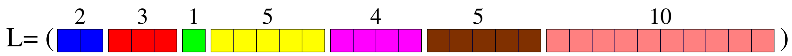
Teorema: ESCALONAMENTO é NP difícil.

Algoritmo de Graham: Escalonar a próxima tarefa na máquina com menos tempo.

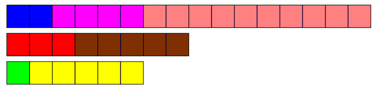
ESCALONAMENTO-GRAHAM (m, n, t)

para j de 1 a m faça $M_j \leftarrow \emptyset$
para i de 1 a n faça
    seja k uma máquina tal que $t(M_k)$ é mínimo
    $M_k \leftarrow M_k \cup \{i\}$
devolva $\{M_1, \dots, M_m\}$

Exemplo:

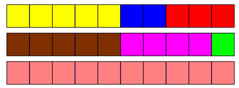


Algoritmo Graham



Graham(L) = 16

Escalonamento Ótimo

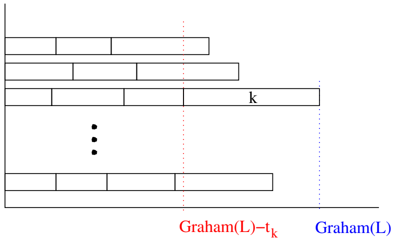


OPT(L) = 10

Teorema: $Graham(L) \leq 2\,\mathrm{OPT}(L)$.

Prova.

Seja J_k a última tarefa terminada no escalonamento.



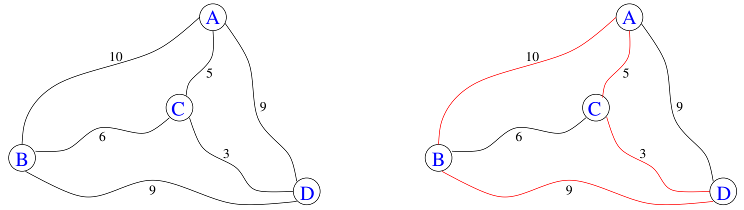
$$\begin{array}{lclcl} \mathrm{OPT}(L) & \geq & \max_i t(J_i) & \geq & t(J_k) \\ \mathrm{OPT}(L) & \geq & \frac{1}{n}\sum_i t(J_i) & \geq & \mathrm{Graham}(L) - t(J_k) \end{array}$$

I.e., $2 \cdot \mathrm{OPT}(L) \geq \mathrm{Graham}(L)$. □

Veja mais sobre algoritmos de aproximação em [CCD+01, Vaz00].

Problema do Caixeiro Viajante

Problema TSP: Dados um grafo G 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 (ex. por um avião).
- Seqüênciamento de DNA (Genoma).
- Cristalografia, seqüenciamento de tarefas.
- Outras: `http://www.math.princeton.edu/tsp/apps`

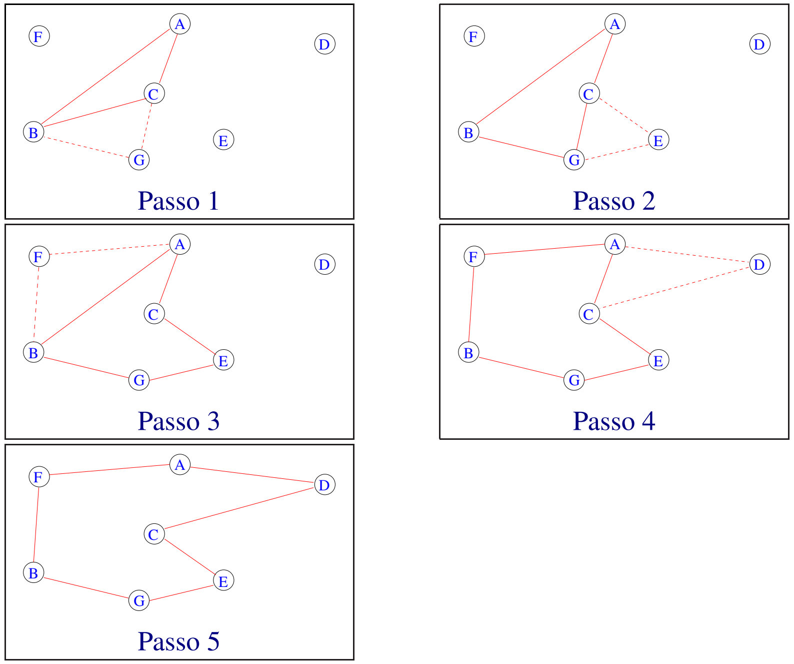
Heurística de Inserção

Estratégia:

- Extendendo circuito com vértice mais "próximo"
- Proximidade de vértice através do peso das arestas para sua adição

INSERÇÃO $(G = (V, E, c))$

```
Escolha um circuito C inicial
Enquanto C não é hamiltoniano faça
    Seja (u,v) ∈ C e x ∉ C tal que c(u,x)+c(x,v) é mínimo
    C ← C − (u,v) + (u,x) + (x,v)
devolva C
```

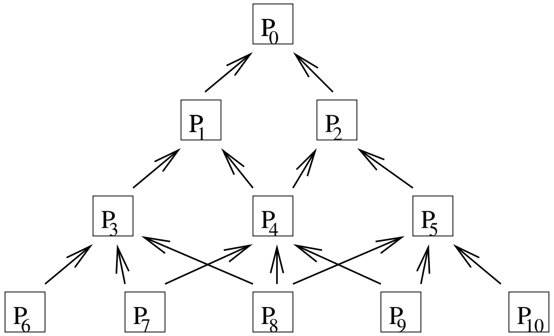



Exercícios:

- Considere o seguinte problema:
  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 $\sum_{i \in S'} s_i \leq B$.
  Descreva um algoritmo guloso para o problema da mochila.
  Sug.: Considere custos relativos.

Programação Dinâmica

- **Construção de soluções através de soluções de subproblemas**
- **Projeto Indutivo**
  – Resolução do problema através de subproblemas menores
- **Estratégia Bottom Up**
  – Resolução dos problemas menores primeiro
- **Exemplos**
  – Problema da Mochila
  – Árvore de Busca Ótima
- **Quando usar Programação Dinâmica ?**
  – Sub-estrutura ótima:
    Uma solução ótima contém soluções ótimas de subproblemas.
  – Subproblemas repetidos:
    Problemas diferentes contém subproblemas iguais.



- Divisão e Conquista/Recursão:
  P_4 resolvido 2 vezes (uma por P_1 e uma por P_2);
  P_7 resolvido 3 vezes (uma por P_3 e duas por P_4);
  P_8 resolvido 4 vezes (uma por P_3, duas por P_4 e uma por P_5);
  P_9 resolvido 3 vezes (uma por P_3 e duas por P_4);
- Programação Dinâmica:
  Cada subproblema resolvido apenas uma vez.

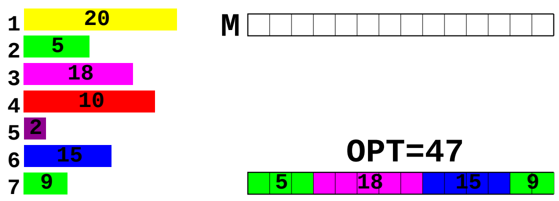
Principais Passos

1. Caracterizar a estrutura de uma solução ótima
2. Recursivamente definir o valor de uma solução ótima
3. Computar o valor de uma solução ótima, de forma bottom-up.
4. Construir uma solução ótima a partir de informações já computadas.

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 $\sum_{i \in S'} s_i \leq B$.

Mochila de capacidade 14

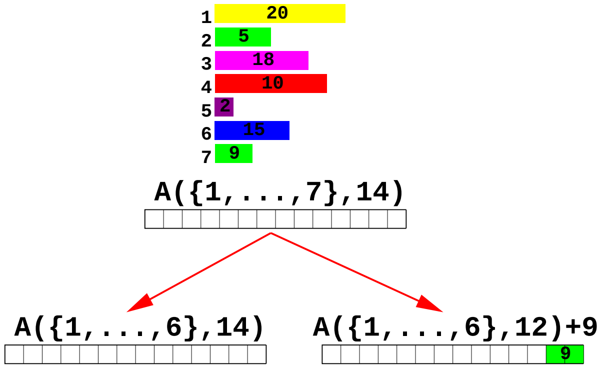


Aplicações:

- Seleção de projetos:
  Uma empresa pode investir até B reais em diferentes projetos $P = \{1, \ldots, n\}$, cada um necessita de um investimento s_i e tem um lucro v_i. Escolher os projetos que maximizam lucro total, e não ultrapassam limite do orçamento.
- Criptografia Publica
  Diffe e Hellman usam sistema de criptografia pública onde a segurança se baseia na dificuldade de se encontrar uma solução para o problema da Mochila.
- Problemas de Empacotamento
  Gilmore e Gomory usam o problema da mochila para resolver problemas de empacotamento (corte de barras, empacotamento em contêineres de carga, etc).

Def.: *Seja $A(\{1, \ldots, i\}, k) \equiv$ o maior valor obtido para um problema da mochila de capacidade k e itens $\{1, \ldots, i\}$*

Como definir soluções ótimas de maneira recursiva ?



$$A(\{1,\ldots,i\},k) = \begin{cases} \max\Big(A(\{1,\ldots,i-1\},k)\ ,\ v_i + A(\{1,\ldots,i-1\},k-s_i)\Big) & \text{se } s_i \leq k \\ A(\{1,\ldots,i-1\},k) & \text{C. c.} \end{cases}$$

$A(\{\},k) = 0$ para todo k

Itens\Mochila	0	1	2	...	$k-s_i$	...	k	...	K
$\emptyset$	0	0	0	...	0	...	0	...	0
$1\ldots1$	0			...		...		...	
$1\ldots2$	0			...		...		...	
$\vdots$	$\vdots$			...		...		...	
$1\ldots i-1$	0			...	$A(i-1,k-s_i)$	...	$A(i-1,k)$	...	
$1\ldots i$	0			...			$A(i,k)$	...	
$\vdots$	$\vdots$			...		...		...	
$1\ldots n$	0			...		...		...	

$$A(i,k) = \begin{cases} \max((A(i-1,k)\ ,\ v_i + A(i-1,k-s_i)) & \text{se } s_i \leq k \\ A(i-1,k) & \text{C. c.} \end{cases}$$

ALGORITMO MOCHILA-EXATA(n, s, v, K)

```
para i ← 0 até n faça A(i,0) ← 0
para k ← 1 até K faça A(0,k) ← 0
para i ← 1 até n faça
    para k ← 1 até K faça
        se s_i ≤ k então
            A(i,k) ← max(A(i−1,k) , v_i + A(i−1,k−s_i))
        senão
            A(i,k) ← A(i−1,k)
```

Solução em: $A(n, K)$

Teorema: *O algoritmo* MOCHILA-EXATA *resolve o problema da mochila em tempo* $O(nK)$.

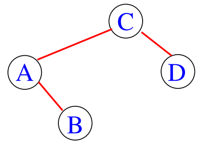
Problema da Árvore de Busca Ótima

Problema ÁRVORE DE BUSCA: Dados elementos $(e_1 \leq e_2 \leq \ldots \leq e_n)$, onde cada item e_i é consultado $f(e_i)$ vezes, construir uma árvore de busca binária, tal que o total de nós consultados é mínimo.

Aplicações: Construção de dicionários estáticos, processadores de texto, verificadores de ortografia.

Exemplo: Considere quatro chaves: $A \leq B \leq C \leq D$

e freqüências $f(A) = 45$, $f(B) = 25$, $f(C) = 18$ e $f(D) = 12$.

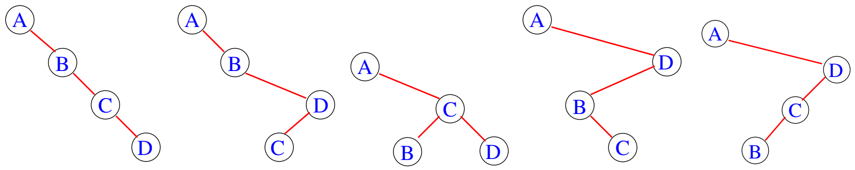


90+75+18+24=207

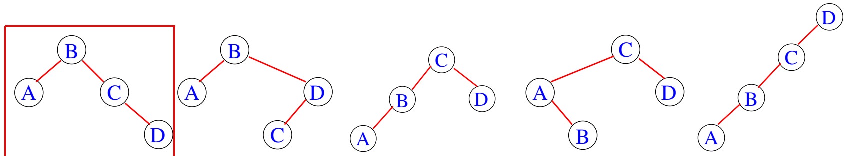
Total de nós acessados nesta árvore = 207

Podemos construir todas as árvores de busca e escolher a melhor ?

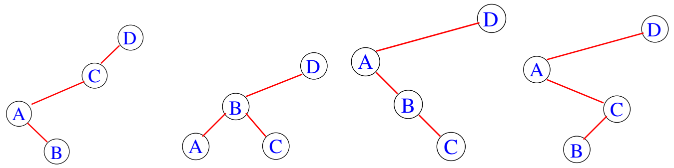
Número de árvores de busca pode ficar muito muito grande!!



45+50+54+48=197 45+50+72+36=203 45+75+36+36=192 45+75+72+24=216 45+100+54+24=223



90+25+36+36=187 90+25+54+24=193 135+50+18+24=227 90+75+18+24=207 180+75+36+12=303



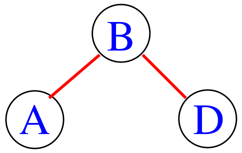
135+100+36+12=283 135+50+54+12=251 90+75+72+12=249 90+100+54+12=256

Propriedades da árvore de busca ótima

Def.: *Se T é uma árvore binária de busca e v é um vértice, denotamos por $T(v)$ a subárvore enraizada em v contendo todos os vértices abaixo de v.*

No exemplo anterior temos $A \leq B \leq C \leq D$.

Pergunta: Em uma árvore de busca T, podemos ter $T(B)$ nesta forma ?

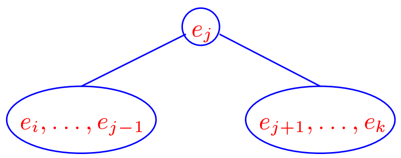


Não: Pois C deveria estar em $T(B)$.

Conclusão: Se T é uma árvore de busca, e v é um vértice de T, então $T(v)$ contém apenas elementos consecutivos.

Seja T uma árvore de busca, e_j um vértice de T e $T(e_j) = \{e_i, \dots, e_{j-1}, e_j, e_{j+1}, \dots, e_k\}$. Então

- no ramo esquerdo devem haver os elementos $e_i, \dots, e_{j-1}$.
- no ramo direito devem haver os elementos $e_{j+1}, \dots, e_k$.
- Sub-árvores de $\{e_i, \dots, e_{j-1}\}$ e $\{e_{j+1}, \dots, e_k\}$ devem ser ótimas.



- Freqüência de acessos à raiz em uma árvore de busca é a soma das freqüências dos elementos na árvore
- Qualquer elemento de $\{e_i, \dots, e_k\}$ é candidato a ser raiz destes

Definição recursiva da solução ótima

Idéia: Gerar árvores de busca a partir de árvores de busca de tamanhos menores

Estratégia: Bottom-Up

Seja
$A(e_i, \dots, e_k)$:=número de nós acessados em árvore ótima contendo $\{e_i, \dots, e_k\}$.

Então

- $A(\emptyset) = 0$
- $A(e_i, \dots, e_k) = \min_{i \le j \le k} \left\{ A(e_i, \dots, e_{j-1}) + A(e_{j+1}, \dots, e_k) + \sum_{t=i}^{k} f(e_t) \right\}$

Item \ Tam.Seq.	0	1	...	t	...	n
e_1	0	$f(e_1)$	...		...	$M(1,n)$
e_2	0	$f(e_2)$	...		...	
$\vdots$	$\vdots$	$\vdots$	...		...	
e_i	0	$f(e_i)$	...	$M(i,t) := A(e_i, \dots, e_k)$	...	
$\vdots$	0	$\vdots$	...		...	
$(k=i+t-1)$ e_k	0	$f(e_k)$	...			
$\vdots$	0	$\vdots$	...		...	
e_n	0	$f(e_n)$	...		...	

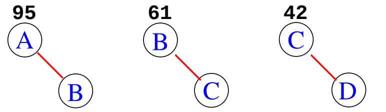
$A(\emptyset) = 0$

$A(e_i) = f(e_i)$

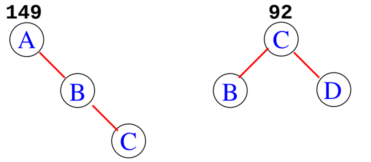
$$A(e_i, \dots, e_k) = \min_{i \le j \le k} \left\{ A(e_i, \dots, e_{j-1}) + A(e_{j+1}, \dots, e_k) + \sum_{t=i}^{k} f(e_t) \right\}$$

Árvores ótimas de tamanho 1

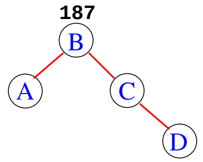
Árvores ótimas de tamanho 2



Árvores ótimas de tamanho 3



Árvores ótimas de tamanho 4



ALGORITMO ÁRVORE-BUSCA$(e_1, \ldots, e_n, f)$

para $i \leftarrow 0$ até $n+1$ faça $M(i,0) \leftarrow 0$
para $t \leftarrow 1$ até n faça
    para $i \leftarrow 1$ até $n-t+1$ faça
        $S \leftarrow f(e_i) + f(e_{i+1}) + \cdots + f(e_{i+t-1})$
        $M(i,t) \leftarrow \min_{0 \le t' \le t-1} \left\{ M(i,t') + M(i+t'+1, t-t'-1) + S \right\}$

Solução em: $A(1,n)$

Teorema: *O algoritmo* ÁRVORE-BUSCA *encontra o valor da árvore de busca ótima em tempo* $O(n^3)$.

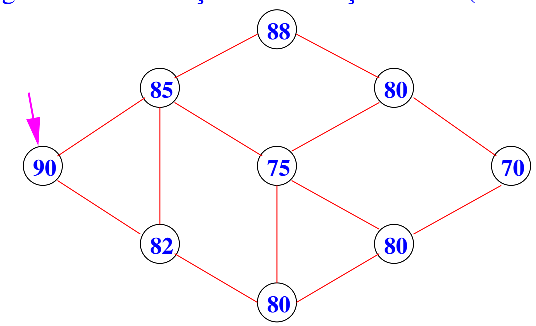
Exercícios:

- O algoritmo apresentado para resolver o problema da mochila apenas apresenta o valor da mochila de peso máximo. Faça uma implementação do algoritmo de maneira que ele apresente também os itens que formam esta solução.
- O algoritmo apresentado para resolver o problema da árvore ótima apenas apresenta o valor esperado de consultas de nós para todos os itens. Faça uma implementação do algoritmo de maneira que ele apresente a árvore de busca ótima.

Heurísticas

Grafo de Vizinhanças

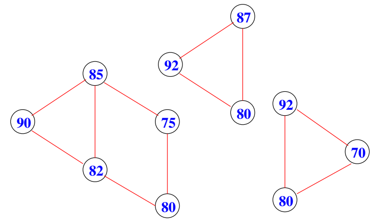
- Vértices são "soluções" viáveis: em geral é um conjunto muito grande
- Arestas indicam transformações de uma solução para outra
- Exemplo de grafo de vizinhança e uma solução inicial (e seu valor)



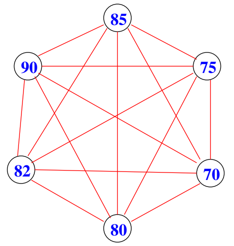
- Objetivo: Chegar na solução de melhor custo pelo grafo de vizinhanças

Grafos de vizinhança ruins

- **Grafo desconexo:** Podemos nunca chegar na solução ótima



- **Grafo denso:** Percorrer os vizinhos de uma "solução" tem a mesma complexidade do problema original



Heurísticas de Busca Local

- **Uso de vizinhança entre soluções viáveis**
- **Uso de solução inicial**
- **Melhorias sucessivas a partir da solução atual**

VIZINHO-MELHOR (S, I)

```
se existe vizinho S' de S com solução melhor
    então retorne S'
    senão retorne ∅
```

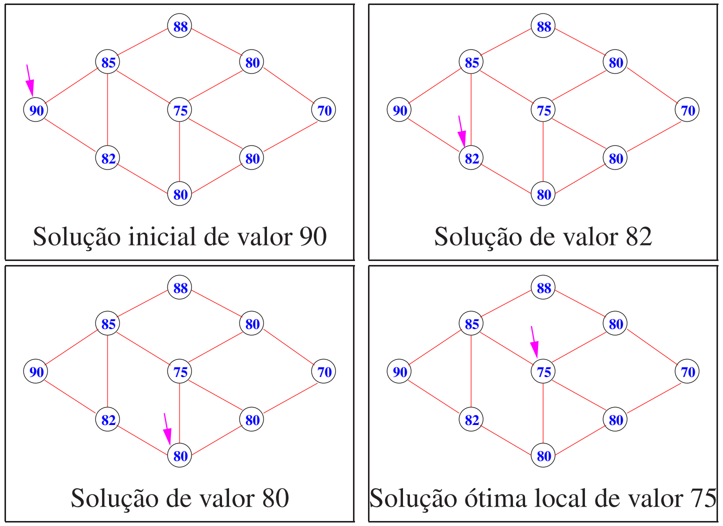
BUSCA-LOCAL-GERAL (I)

```
encontre "solução" inicial S para I
S' ←VIZINHO-MELHOR(S, I)
enquanto S' ≠ ∅ faça
    S ← S'
    S' ←VIZINHO-MELHOR(S, I)
devolva S
```

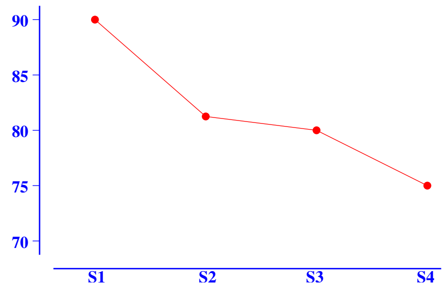

Algoritmo Melhor-Vizinho:

Escolher o melhor dentre todos os vizinhos que tem valor melhor

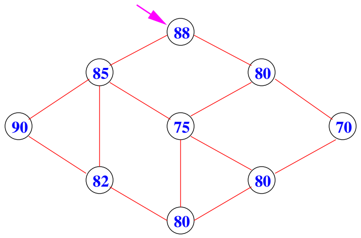
Grafo de vizinhança entre "soluções" viáveis



Comportamento do algoritmo de busca local

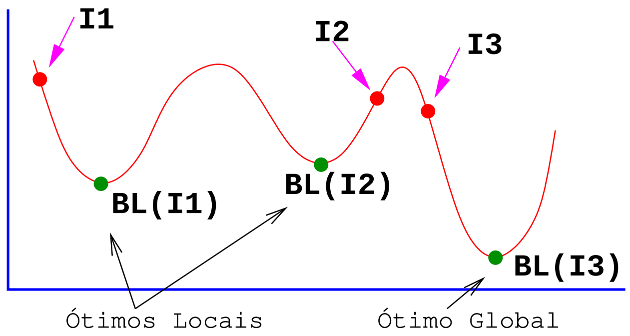


Se a primeira solução fosse a de valor 88, teríamos chegado na solução ótima

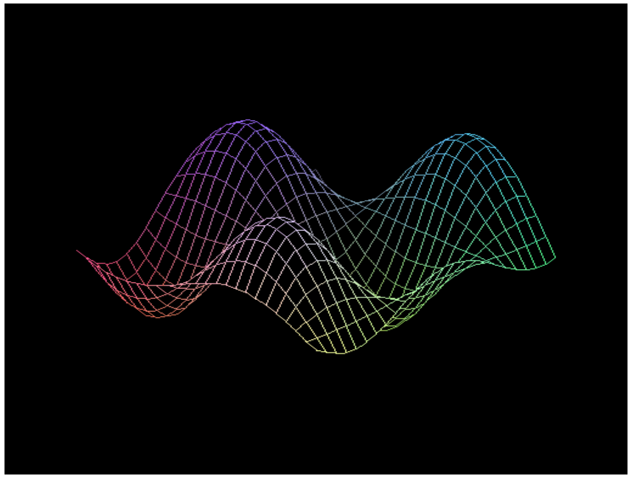


Saindo de mínimos locais: Multi-Start Local Search

- **Executar algoritmo de Busca Local com diferentes inícios**
- **Guardar melhor solução**
- **Exemplo em otimização unidimensional**



Mínimos e máximos locais em função bidimensional



Busca Local para o TSP

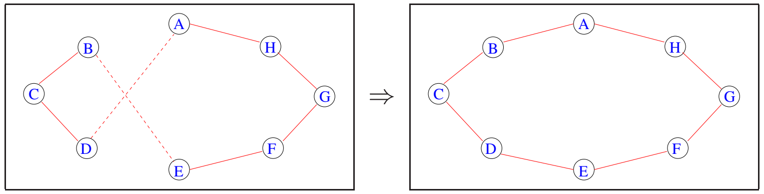
Considere grafos completos

Vizinhança-K-OPT$(C) := \{C' : C'$ é circuito hamiltoniano obtido de C removendo K arestas e inserindo outras K arestas.$\}$.

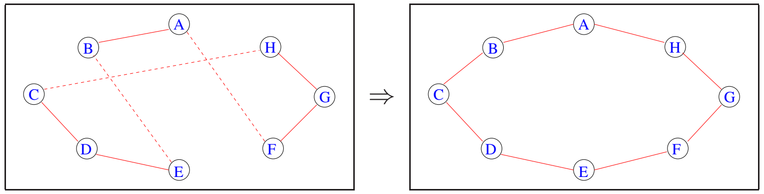
K-OPT $(G = (V, E, c))$

1 encontre um circuito hamiltoniano inicial C
2 repita
3 procure C' em Vizinhança-K-OPT(C) tal que $\mathrm{val}(C') < \mathrm{val}(C)$.
4 se encontrou tal C', $C \leftarrow C'$
5 até que não se consiga encontrar tal C' no passo 3
6 devolva C

Exemplo de troca 2-OPT



Exemplo de troca 3-OPT



Uma solução viável pode ter vários vizinhos usando troca 3-OPT.
Quantos ?

Comparação em grafos euclidianos aleatórios (Johnson&McGeoch'97)

- **Usando algoritmo guloso para obter circuito inicial**
  - Inserindo arestas mais leves primeiro
  - Descartando arestas que inviabilizam solução
- **Comparando com limitante inferior do ótimo**
  - Limitante de Held-Karp

$N =$	10^2	$10^{2.5}$	10^3	$10^{3.5}$	10^4	$10^{4.5}$	10^5	$10^{5.5}$	10^6
Guloso	19.5	18.8	17.0	16.8	16.6	14.7	14.9	14.5	14.2
2-OPT	4.5	4.8	4.9	4.9	5.0	4.8	4.9	4.8	4.9
3-OPT	2.5	2.5	3.1	3.0	3.0	2.9	3.0	2.9	3.0

Fator de excesso em relação ao limitante de Held-Karp

Apêndice I: Grafos

Def.: Um subgrafo $H = (V_H, E_H)$ de $G = (V, E)$ é um grafo tal que $V_H \subseteq V$, $E_H \subseteq E$ e $E_H \subseteq V_H \times V_H$.

Def.: Um subgrafo $H = (V_H, E_H)$ de $G = (V, E)$ é gerador se $V_H = V$.

Def.: Uma passeio P em $G = (V, E)$ é uma seqüência $(v_0, e_1, v_1, e_2, \ldots, v_{k-1}, e_{k-1}, v_k)$ onde $v_i \in V$ e $e_i = \{v_{i-1}, v_i\} \in E$. Chamamos v_0 e v_k de origem e destino de P, resp.

Def.: Uma trilha é um passeio que não repete arestas.

Def.: Uma caminho é um passeio que não repete vértices.

Def.: Um ciclo é uma trilha com os vértices de origem e destino iguais.

Def.: Um ciclo euleriano de G é uma ciclo que contém todas as arestas de G.

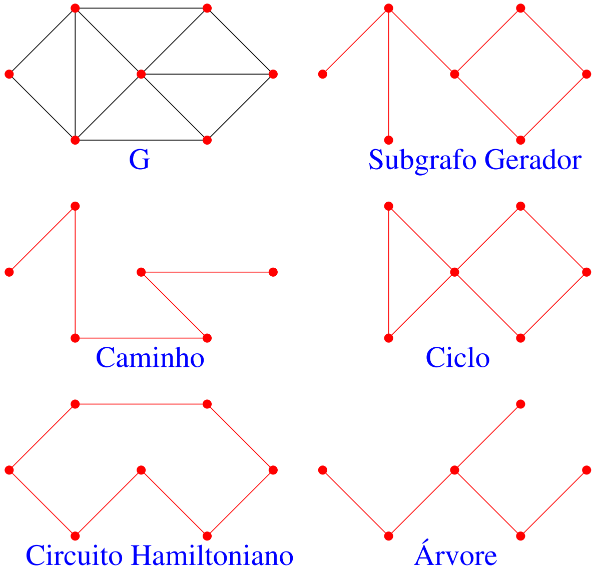
Def.: Um circuito é um ciclo que não repete vértices.

Def.: Um circuito hamiltoniano de G é um circuito que contém todos os vértices de G.

Def.: Um grafo $G = (V, E)$ é conexo se para todo $u, v \in V$, existe caminho ligando u e v.

Def.: Uma floresta $F = (V, E)$ é um grafo que não contém circuitos.

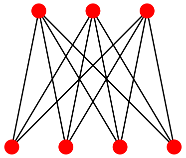
Def.: Uma árvore $F = (V, E)$ é um grafo conexo sem circuitos.



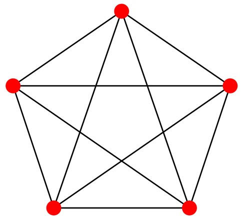
Def.: Um grafo $G = (V, E)$ é bipartido se existe partição (X, Y) de V tal que $E \subseteq X \times Y$.

Def.: Um grafo $G = (V, E)$ é bipartido completo se existe partição (X, Y) de V tal que $E = X \times Y$. Denotado por $K_{|X| \times |Y|}$.

Def.: Um grafo $G = (V, E)$ é completo se $E = V \times V$. Denotado por $K_{|V|}$.



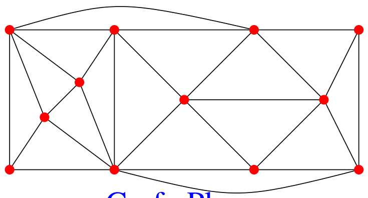
K_{3x4}



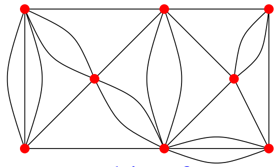
K_5

Def.: Um grafo $G = (V, E)$ é planar se pode ser desenhado no plano com arestas interceptando apenas nos extremos.

Def.: Um multigrafo $G = (V, E)$ é uma extensão de grafos para admitir várias cópias de uma aresta.



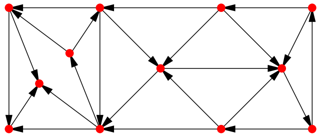
Grafo Planar



Multigrafo

Def.: *Uma aresta orientada é um par ordenado (u, v) representada por $u \to v$ para vértices u e v. Dizemos que u é o início (ou cabeça) e v é o fim (calda) da aresta.*

Def.: Um grafo $G = (V, E)$ é dito ser orientado se suas arestas são orientadas.



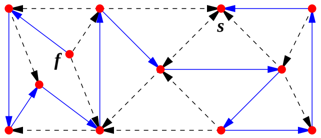
Grafo Orientado

Def.: Um vértice é dito ser fonte se não há arestas entrando no vértice.

Def.: Um vértice é dito ser sorvedouro se não há arestas saindo do vértice.

Algumas definições anteriores se aplicam para grafos orientados.

Exemplo de caminho hamiltoniano orientado de f para s



Caminho hamiltoniano

Def.: *Dado um grafo não orientado $G = (V, E)$ e um vértice $v \in V$, denotamos por $\delta(v)$ o conjunto das arestas que tem um extremo em v.*

Def.: *Dado um grafo não orientado $G = (V, E)$ e conjunto $S \subseteq V$, denotamos por $\delta(S)$ o conjunto das arestas que tem exatamente um extremo em S.*

Def.: *Dado um grafo orientado $G = (V, E)$ e um vértice $v \in V$, denotamos denotamos por $\delta^+(v)$ o conjunto das arestas de E que tem a calda em v e denotamos por $\delta^-(v)$ o conjunto das arestas de E que tem o início em v.*

Def.: *Dado um grafo orientado $G = (V, E)$ e conjunto $S \subseteq V$, denotamos por $\delta^+(S)$ o conjunto das arestas de E que tem a calda em S e o início em $V \setminus S$ e denotamos por $\delta^-(S)$ o conjunto das arestas de E que tem o início em S e a calda em $V \setminus S$.*

Para entender mais sobre grafos, veja Bondy and Murty [BM76].

Apêndice II: Definições e notação sobre conjuntos

Def.: Dado um conjunto A e uma função f sobre A, $f : A \to \mathbb{R}$, e um elemento $e \in A$, denotamos por $f_e = f(e)$.

Def.: Dado um conjunto A e uma função numérica $f : A \to \mathbb{R}$, denotamos por $f(B) := \sum_{e \in B} f(b)$ para todo $B \subseteq A$.

Exemplo: Se $G = (V, E)$ é um grafo, v é um vértice de V, e x é uma função numérica nas arestas $x : E \to \mathbb{R}$ então vale

$$x(\delta(v)) = \sum_{e \in \delta(v)} x_e = \sum_{e \in \delta(v)} x(e)$$

(soma dos valores da função x aplicada nas arestas que incidem em v).

Bibliografia

Referências

[AMO93] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice-Hall, 1993.

[BM76] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. Macmillan/Elsevier, 1976.

[CCD+01] M.H. Carvalho, M.R. Cerioli, R. Dahab, P. Feofiloff, C.G. Fernandes, C.E. Ferreira, K.S. Guimarães, F.K. Miyazawa, J.C. Pina Jr., J. Soares, and Y. Wakabayashi. *Uma introdução sucinta a algoritmos de aproximação*. Editora do IMPA - Instituto de Matemática Pura e Aplicada, Rio de Janeiro–RJ, 2001. M.R. Cerioli, P. Feofiloff, C.G. Fernandes, F.K. Miyazawa (editors).

[Chv83] V. Chvátal. *Linear Programming*. Freeman, 1983.

[CK00] P. Crescenzi and V. Kann. *A Compendium of NP Optimization Problems*, 2000. URL: `http://www.nada.kth.se/~viggo /wwwcompendium/`.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 1990.

[FW96] C.E. Ferreira and Y. Wakabayashi. *Combinatória poliédrica e planos de corte faciais*. X Escola de Computação, UNICAMP, Campinas–SP, 1996.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-completeness*. W. H. Freeman, San Francisco, CA, 1979.

[Kar72] R.M. Karp. Reducibility among combinatorial problems. In

R.E. Miller and J.M. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, 1972.

[LLRS90] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley, 1990. Reprint of the 1985 original.

[Pap94] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, 1994.

[PS82] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, 1982.

[Vaz00] V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2000.

[Wol98] L.A. Wolsey. *Integer Programming*. Wiley, 1998.