Teoria da Complexidade

Cid C. de Souza / IC–UNICAMP

Universidade Estadual de Campinas
Instituto de Computação

1º semestre de 2012

Revisado por Zanoni Dias

Autor

Prof. Cid Carvalho de Souza
Universidade Estadual de Campinas (UNICAMP)
Instituto de Computação
Av. Albert Einstein nº 1251
Cidade Universitária Zeferino Vaz
13083-852, Campinas, SP, Brasil
Email: cid@ic.unicamp.br

Direitos autorais

- Este material só pode ser reproduzido com a autorização do autor.
- Os alunos dos cursos do Instituto de Computação da UNICAMP bem como os seus docentes estão autorizados (e são bem vindos) a fazer uma cópia deste material para estudo individual ou para preparação de aulas a serem ministradas nos cursos do IC/UNICAMP.
- Se você tem interesse em reproduzir este material e não se encontra no caso acima, por favor entre em contato comigo.
- Críticas e sugestões são muito bem vindas!

Campinas, agosto de 2010.

Cid

Tratamento de problemas $\mathcal{NP}$-difíceis: *branch & bound*.

▷ Casos de aplicação:
- Problemas cujas soluções podem ser representadas por tuplas (vetores) de tamanho fixo ou variável da forma $(x_1, \ldots, x_n)$.
- Solucionar o problema equivale a encontrar uma tupla que otimiza uma *função critério* $P(x_1, \ldots, x_n)$.

▷ Restrições:
- *Explícitas*: especificam os domínios (finitos) das variáveis na tupla.
- *Implícitas*: relações entre as variáveis da tupla que especificam quais delas respondem ao problema.

Branch & Bound: conceitos básicos

- ▷ **Espaço de soluções**: conjunto de todas as tuplas satisfazendo as restrições *explícitas*.
- ▷ **Espaço de estados**: conjunto de todas as subseqüências das tuplas do *espaço de soluções*.
- ▷ Métodos de exploração do espaço de estados (EE):
  - *nós ativos*: aqueles que ainda têm filhos a serem gerados.
  - *nós amadurecidos*: aqueles em que todos os filhos já foram gerados ou não devam ser mais expandidos de acordo com a *função limitante*.
  - *nó corrente*: aquele que está sendo explorado.

Tratamento de problemas $\mathcal{NP}$-difíceis: *branch & bound*

- ▷ Exploração do espaço de estados: todos os filhos de um nó da árvore de espaço de estados são gerados ao mesmo tempo.
- ▷ Estratégia do melhor limitante (*best bound*): próximo nó a ser explorado é indicado por uma *função classificadora*.
- ▷ Em cada nó da árvore, a *função classificadora* estima o melhor valor da função objetivo no subespaço das soluções representadas por aquele nó.
- ▷ Os nós são *amadurecidos* por: **(1)** inviabilidade (não satisfazer as restrições implícitas); **(2)** limitante (função classificadora indica que ótimo não pode ser atingido naquela subárvore) ou **(3)** otimalidade (ótimo da subárvore já foi encontrado).

Algoritmo genérico de *Branch & Bound*

B&B(k); (* considerando problema de **maximização** *)
    Nós-ativos $\leftarrow$ {nó raiz}; melhor-solução $\leftarrow$ {}; $\underline{z} \leftarrow -\infty$;
    Enquanto (Nós-ativos não está vazia) **faça**
        Escolher um nó k em Nós-ativos para ramificar;
        Remover k de Nós-ativos;
        Gerar os filhos de k: $n_1, \ldots, n_q$ e computar os $\overline{z}_{n_i}$ correspondentes;
            (* $\overline{z}_{n_i} \leftarrow -\infty$ se restrições implícitas não são satisfeitas *)
        Para $i = 1$ **até** q **faça**
            se ($\overline{z}_{n_i} \leq \underline{z}$) **então** amadurecer o nó n_i;
            se não
                Se (n_i representa uma solução completa) **então**
                    $\underline{z} \leftarrow \overline{z}_{n_i}$; melhor-solução $\leftarrow$ {solução de n_i};
                se não adicionar n_i à lista Nós-ativos.
            fim-se
        fim-para
    fim-enquanto
fim.

Branch & Bound: mochila binária (BKP)

- ▷ Dados n itens com pesos positivos $w_1, \ldots, w_n$ e valores positivos $c_1, \ldots, c_n$, encontrar um subconjunto de itens de **valor máximo** e cujo peso não exceda a capacidade da mochila dada por um valor positivo W.
- ▷ *Função classificadora*: como estimar o valor da *função objetivo*?
- ▷ *Relaxação*: posso levar qualquer fração de um item.
- ▷ Algoritmo para o problema relaxado quando os itens estão ordenados de forma que $\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq \ldots \geq \frac{c_n}{w_n}$.
- ▷ *Por quê funciona?*

```
Calcula_z̄(W, C, k); (* função classificadora para BKP *)
    j ← k + 1;
    W' ← W;
    C' ← C;
    Enquanto W' ≠ 0 faça
        x_j ← min{W'/w_j, 1};
        W' ← W' − w_j x_j;
        C' ← C' + c_j x_j;
        j ← j + 1;
    enquanto
    Retornar C';
fim
```


Branch & Bound: mochila binária (BKP)

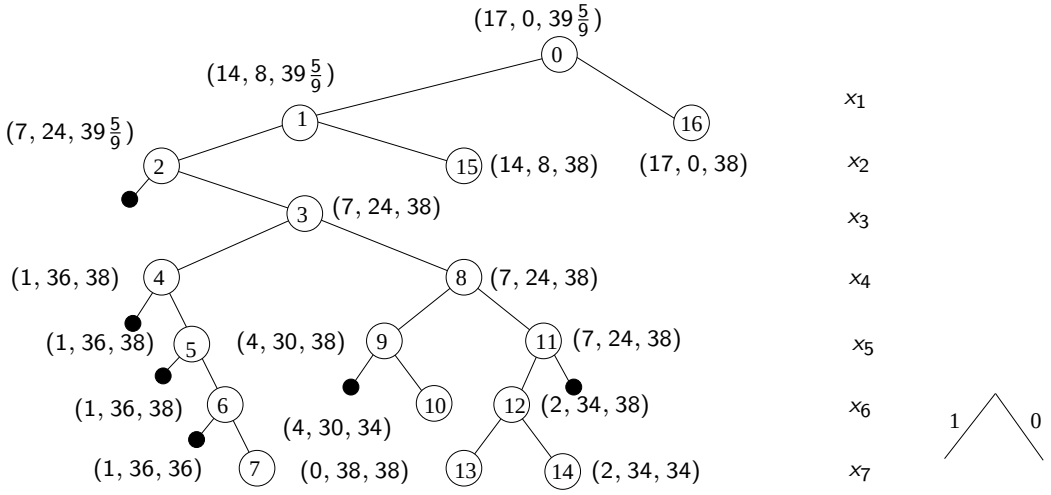
- ▷ Exemplo:

$$\begin{array}{ll} \max & 8x_1 + 16x_2 + 20x_3 + 12x_4 + 6x_5 + 10x_6 + 4x_7 \\ & 3x_1 + 7x_2 + 9x_3 + 6x_4 + 3x_5 + 5x_6 + 2x_7 \leq 17 \\ & x_i \in \{0,1\}, i = 1, \ldots, n. \end{array}$$

- ▷ Parte explorada da árvore de espaço de estados (próxima transparência).
- ▷ Legenda: $(W', C, \overline{z}_{n_i})$ onde W' é a capacidade restante na mochila, C é o custo da solução parcial correspondente ao nó e $\overline{z}_{n_i}$ é o valor do limitante obtido pela função classificadora no nó.

C. de Souza Teoria da Complexidade

Branch & Bound: problema da mochila (cont.)



Ordem de geração dos nós: $0, 1, 16, 2, 15, 3, 4, 8, 5, 6, 7, 9, 11, 10, 12, 13, 14$

Branch & Bound: *Flowshop Scheduling* (FSP)

- ▷ **Dados de entrada**: conjunto de n tarefas $J_1, \ldots, J_n$ cada uma delas composta de duas subtarefas sendo que a primeira deve ser executada na máquina 1 e a segunda na máquina 2, somente após encerrada a execução da primeira. O tempo de processamento da tarefa J_j na máquina i é dado por t_{ij}.
- ▷ **Definição**: o *tempo de término* da tarefa J_j na máquina i é dado por f_{ij}.
- ▷ **Pede-se**: encontrar uma seqüência de execução das subtarefas nas máquinas de modo que a soma dos tempos de término na máquina 2 seja mínima. Ou seja, a função objetivo é:

$$\min f = \sum_{j=1}^{n} f_{2j}.$$

Branch & Bound: FSP (cont.)

▷ **Resultados conhecidos para o FSP**:

- a versão de decisão de FSP é $\mathcal{NP}$-completo.
- Existe um escalonamento ótimo no qual a seqüência de execução das tarefas é a mesma nas duas máquinas (**permutation schedules**) e no qual não há tempo ocioso *desnecessário* entre as tarefas.

▷ Exemplo: $n = 3$.

t_{ij}	Máquina 1	Máquina 2
Tarefa 1	2	1
Tarefa 2	3	1
Tarefa 3	2	3

Branch & Bound: (FSP) (cont.)

▷ *Permutation Schedule* ótimo: $f = 18$

Máquina 1	1	1	3	3	2	2	2	
Máquina 2			1		3	3	3	2

▷ Outro *Permutation Schedule*: $f = 21$

Máquina 1	2	2	2	3	3	1	1	
Máquina 2				2		3	3	3	1

Branch & Bound: FSP (cont.)

- ▷ Representação da solução: como existe uma soluçao ótima que é um *permutation schedule*, o natural seria utilizar uma tupla $(x_1, \ldots, x_n)$ de tamanho fixo onde x_i é o número da i-ésima tarefa da permutação.
- ▷ Suponha que num dado nó da árvore as tarefas de um subconjunto M de tamanho r tenham sido escalonadas. Seja t_k, $k = 1, \ldots, n$, o índice da k-ésima tarefa em qualquer escalonamento que possa ser representado por um nó na subárvore cuja raiz é o nó corrente. O custo deste escalonamento será:

$$f = \sum_{i \in M} f_{2i} + \sum_{i \notin M} f_{2i}.$$

Branch & Bound: FSP (cont.)

- ▷ Como o primeiro termo da soma já está definido, as seguintes *funções classificadoras* poderiam estimar o valor do outro termo:

$$S_1 = \sum_{k=r+1}^{n} [f_{1,t_r} + (n-k+1)t_{1,t_k} + t_{2,t_k}],$$

na qual assume-se que cada tarefa começa a ser executada na máquina 2 imediatamente após a sua conclusão na máquina 1, e

$$S_2 = \sum_{k=r+1}^{n} [\max(f_{2,t_r}, f_{1,t_r} + \min_{i \notin M} t_{1i}) + (n-k+1)t_{2,t_k}],$$

na qual assume-se que cada tarefa começa na máquina 2 imediatamente depois que a tarefa precedente termina sua execução na máquina 2.

C. de Souza Teoria da Complexidade

Branch & Bound: FSP (cont.)

- ▷ A minimização de S_1 pode ser obtida ordenando-se as tarefas na ordem crescente dos valores de t_{1,t_k}.
- ▷ A minimização de S_2 pode ser obtida ordenando-se as tarefas na ordem crescente dos valores de t_{2,t_k}.
- ▷ Se $\hat{S}_1$ e $\hat{S}_2$ são os mínimos acima, tem-se um *limitante inferior* facilmente calculado por:

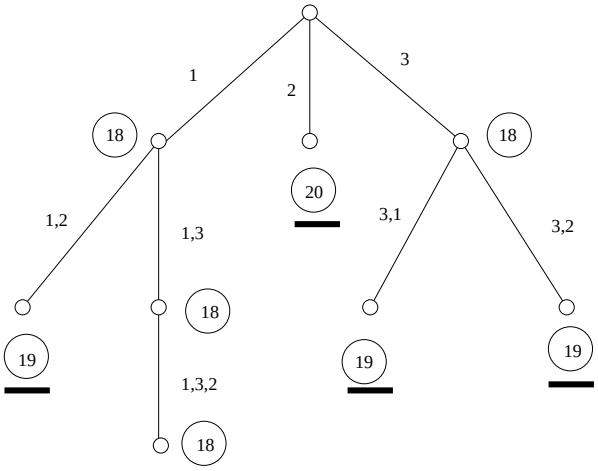
$$f \geq \sum_{i \in M} f_{2i} + \max(\hat{S}_1, \hat{S}_2).$$

Branch & Bound: FSP (cont.)

- ▷ Exemplo (continuação): os valores computados para estimar f para os três nós filhos da raiz seriam:

$$f = \begin{cases} 18 & \text{se a tarefa 1 for escalonada primeiro;} \\ 20 & \text{se a tarefa 2 for escalonada primeiro;} \\ 18 & \text{se a tarefa 3 for escalonada primeiro.} \end{cases}$$

- ▷ Parte da árvore de espaços gerada: próxima transparência.



Tratamento de problemas $\mathcal{NP}$-difíceis: Programação Linear Inteira (PLI)

▷ **Programação Linear Inteira** (PLI):

Problema PLI *Puro*:

$$\begin{array}{rrcl} \min & z = cx & & \\ \text{Sujeito a} & Ax & \leq & b \\ & x & \in & \mathbb{Z}^n_+ \end{array}$$

Problema PLI *Misto*:

$$\begin{array}{rrcl} \min & z = cx + dy & & \\ \text{Sujeito a} & Ax + By & \leq & b \\ & x & \in & \mathbb{R}^p_+ \\ & y & \in & \mathbb{Z}^n_+ \end{array}$$

▷ Versão de *decisão*: dada uma matriz inteira $A : m \times n$, dois vetores inteiros $c : 1 \times n$ e $b : m \times 1$ e um valor inteiro q, determinar se existe $x \in \mathbb{Z}^n$ tal que $Ax \leq b, x \geq 0$ e $cx \leq q$.

Programação Linear Inteira (PLI)

- ▷ *Teorema*: PLI $\in \mathcal{NP}$.
  É possível provar que se o sistema tem solução então existe uma solução (vetor) cujo valor de cada componente é limitado polinomialmente pelo tamanho entrada, ou seja, existe um certificado sucinto para PLI.
- ▷ *Teorema*: PLI $\in \mathcal{NP}$-difícil.
  Basta provar que um problema de $\mathcal{NP}$-difícil se reduz polinomialmente a PLI. Mas isso é **equivalente a formular o problema usando programação linear inteira***!*

Programação Linear Inteira: formulações

- ▷ Exemplo 1: SAT $\propto_{\text{poli}}$ PLI.
- ▷ Instância do SAT: $\mathcal{F} = (x + y + \overline{z}).(\overline{x} + \overline{y} + z).(y + \overline{z})$ com $m = 3$ cláusulas e $n = 3$ variáveis.
- ▷ Formulação PLI: criar 6 variáveis binárias $x, y, z, \overline{x}, \overline{y}$ e $\overline{z}$ que terão valor *um* se as literais correspondentes na fórmula $\mathcal{F}$ forem *verdadeiras* e terão valor *zero* caso contrário.

$$
\begin{array}{rlllllll}
\min & w = x & \text{(* qualquer função linear serve! *)} & & & & & \\
\text{Sujeito a} & x + y + \overline{z} & \geq & 1, & \overline{x} + \overline{y} + z & \geq & 1, \\
 & y + \overline{z} & \geq & 1, & x + \overline{x} & = & 1, \\
 & y + \overline{y} & = & 1, & z + \overline{z} & = & 1, \\
 & x, y, z, \overline{x}, \overline{y}, \overline{z} \in \{0, 1\} & & & & & &
\end{array}
$$

Programação Linear Inteira: formulações (cont.)

- ▷ Exemplo 2: CLIQUE $\propto_{\text{poli}}$ PLI.
- ▷ Instância de CLIQUE: grafo $G = (V, E)$ com $|V| = n$ e $|E| = m$.
- ▷ Formulação PLI: para cada vértice $u \in V$ cria-se uma variável binária x_u que vale *um* se e somente se o vértice u está na clique.
- ▷ Função objetivo (CLIQUE de maior tamanho): $\max \sum_{u \in V} x_u$.
- ▷ Restrições: para cada aresta (u, v) que não está em E pelo menos um dos vértices não pode estar na clique, ou seja, $x_u + x_v \leq 1$.
- ▷ Logo a formulação se reduz a:

$$\begin{array}{rlll} \max & z = \sum_{u \in V} x_u & & \\ \text{Sujeito a} & x_u + x_v & \leq 1, & \forall (u, v) \notin E \\ & x_u & \in \{0, 1\} & \forall u \in V. \end{array}$$

Programação Linear Inteira: formulações (cont.)

- ▷ Exemplo 3 (cobertura de vértices): CV $\propto_{\text{poli}}$ PLI.
- ▷ Instância de CV: grafo $G = (V, E)$ com n vértices e m arestas.
- ▷ Formulação PLI: para cada $u \in V$ cria-se uma variável binária x_u que vale *um* se e somente se o vértice u está na cobertura.
- ▷ Função objetivo (cobertura de menor tamanho): $\min \sum_{u \in V} x_u$.
- ▷ Restrições: para cada aresta (u, v) de E pelo menos um dos seus vértices extremos está na cobertura, ou seja, $x_u + x_v \geq 1$.
- ▷ Logo a formulação se reduz a:

$$\begin{array}{rlll} \min & z = \sum_{u \in V} x_u & & \\ \text{Sujeito a} & x_u + x_v & \geq 1, & \forall (u, v) \in E \\ & x_u & \in \{0, 1\} & \forall u \in V. \end{array}$$

Programação Linear Inteira: formulações (cont.)

- ▷ Exemplo 4 (3 coloração): 3COLOR $\propto_{\text{poli}}$ PLI.
- ▷ Instância de 3COLOR: grafo $G = (V, E)$ com $|V| = n$ e $|E| = m$.
- ▷ Formulação PLI (variáveis):
  - uma variável binária x_{uk} para cada vértice $u \in V$ e cada cor $k \in \{1, 2, 3\}$ tal que $x_{uk} = 1$ *se e somente se* o vértice u foi colorido com a cor k.
  - uma variável binária y_k para toda cor $k \in \{1, 2, 3\}$ cujo valor será *um* se e somente se algum vértice receber a cor k.
- ▷ Função objetivo (minimizar o número de cores usadas):

$$\min \sum_{k=1}^{3} y_k.$$

Programação Linear Inteira: formulações (cont.)

- ▷ Restrições (3COLOR):
  - Todo vértice deve receber exatamente uma cor, ou seja,

$$\sum_{k=1}^{3} x_{uk} = 1, \forall u \in V.$$

  - Se um vértice recebe uma cor k, esta cor tem que ser usada:

$$x_{uk} \leq y_k, \forall u \in V, k = 1, 2, 3.$$

  - Uma cor só pode ser usada se algum vértice tiver aquela cor:

$$y_k \leq \sum_{u \in V} x_{uk}, k = 1, 2, 3.$$

  - Os vértices extremos de uma aresta não podem ter a mesma cor: $x_{uk} + x_{vk} \leq 1, \forall (u, v) \in E, k = 1, 2, 3.$

Programação Linear Inteira: formulações (cont.)

- ▷ Exemplo 5 (*Scheduling* com janela de tempo): SJT $\propto_{\text{poli}}$ PLI.
- ▷ Instância de SJT: um conjunto T de n tarefas e, para cada $t \in T$, um prazo de início r_t, uma duração ℓ_t e um prazo de conclusão d_t, sendo r_t, d_t e ℓ_t inteiros não-negativos. Decidir se existe um *seqüenciamento viável* das tarefas de T em uma máquina.
- ▷ Variáveis naturais: para todo $t \in T$ o instante de início de execução da tarefa é dado por σ_t.
- ▷ Função objetivo: qualquer função linear serve, e.g., $\min \sigma_1$.
- ▷ Restrições envolvendo um única tarefa:

$$\begin{array}{lcll} \sigma_t & \geq & r_t, & \forall t \in T \quad \text{(início da tarefa)} \\ \sigma_t + \ell_t & \leq & d_t, , & \forall t \in T \quad \text{(fim da tarefa)} \end{array}$$

Programação Linear Inteira: SJT (cont.)

- ▷ Variáveis binárias: para poder representar corretamente a relação entre os tempos de início de duas tarefas t e t' em T é necessário que se saiba qual delas irá ser executada antes.

$$y_{tt'} = \begin{cases} 1, \text{se } t \text{ antecede } t' \\ 0, \text{caso contrário} \end{cases}, \text{ para todo par } \{t, t'\} \in T.$$

- ▷ Restrições envolvendo pares de tarefas:

$$\begin{array}{lcll} y_{tt'} + y_{t't} & = & 1, & \forall \{t, t'\} \in T \\ \sigma_t + \ell_t & \leq & \sigma_{t'} + (1 - y_{tt'})M, & \forall \{t, t'\} \in T \\ \sigma_{t'} + \ell_{t'} & \leq & \sigma_t + y_{tt'}M, & \forall \{t, t'\} \in T \end{array}$$

onde M é um valor suficientemente grande. Por exemplo, M poderia ser $\max_{t \in T}\{d_t\} - \min_{t \in T}\{r_t\}$.

Programação Linear Inteira: formulações (cont.)

▷ Exemplo 5 (Problema de Transporte): uma grande empresa de consultoria possui m escritórios e n clientes espalhados em todo Brasil. No escritório i estão baseados a_i consultores e cada cliente j, para $j = 1, \ldots, n$, contratou b_j consultores. O custo de deslocar um consultor do escritório i para o cliente j é c_{ij}.
Equacionar este problema como um PLI.

▷ Variáveis: para todo par (escritório i, cliente j), define-se a variável **inteira** x_{ij} que representa o número de consultores que serão deslocados do escritório i para o cliente j.

Programação Linear Inteira: formulações (cont.)

▷ A formulação do problema como um PLI é dada por:

$$
\begin{aligned}
\text{min} \quad & z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \\
\text{Sujeito a} \quad & \sum_{j=1}^{n} x_{ij} \leq a_i, \quad i = 1, \ldots, m \\
& \sum_{i=1}^{m} x_{ij} = b_j, \quad j = 1, \ldots, n \\
& x_{ij} \in \mathbb{Z}_+^n, \quad i = 1, \ldots, m \text{ e } j = 1, \ldots, n.
\end{aligned}
$$

▷ *Observação*: este problema pode ser resolvido em tempo polinomial!

Programação Linear Inteira: formulações (cont.)

▷ Exemplo 6 (Problema de Localização de Facilidades **Não Capacitado** (UFL)): Dado um conjunto $N = \{1, \ldots, n\}$ de locais potenciais para instalação de depósitos e um conjunto $M = \{1, \ldots, m\}$ de clientes, suponha que f_j seja o custo de instalar o depósito em j e que c_{ij} seja o custo de transportar toda demanda de mercadorias do depósito j para o cliente i. Decidir quais depósitos instalar e que fração da demanda de cada cliente deve ser atendida por cada depósito.

▷ Variáveis:

$$y_j = \begin{cases} 1, \text{se for instalado um depósito em } j \\ 0, \text{caso contrário} \end{cases}$$

Programação Linear Inteira: formulações (cont.)

▷ Variáveis (cont.):

$x_{ij} \in [0, 1]$: fração da demanda do cliente i atendida pelo depósito j.

▷ Restrições:

- satisfação da demanda: $\sum_{j \in N} x_{ij} = 1, \forall\, i \in M.$
- uso do depósito j: $\sum_{i \in M} x_{ij} \leq m y_j, \forall\, j \in N.$

▷ Função objetivo: $\min z = \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} + \sum_{j \in N} f_j y_j.$

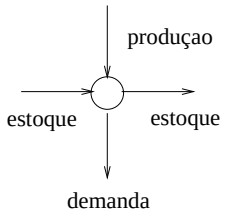
Programação Linear Inteira: formulações (cont.)

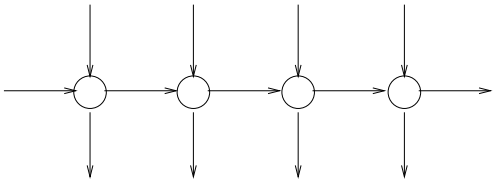
▷ Exemplo 7 (Problema do planejamento da produção **capacitado** (CLS)): decidir as quantidades a produzir de um certo produto em um horizonte de planejamento de n períodos de tempo. Os dados de entrada são:

f_t: **custo fixo** de produção no período t;
p_t: custo unitário de produção no período t;
h_t: custo unitário de estocagem no período t;
d_t: demanda no período t;
C_t: a capacidade de produçao no período t;
s_0, s_n: os estoques inicial e final do produto.

Programação Linear Inteira: formulações (cont.)

▷ Um modelo gráfico:





▷ Variáveis:

x_t: quantidade produzida no período t;
s_t: estoque no período t;
y_t : $\begin{cases} 1, & \text{se for decidido produzir no período } t; \\ 0, & \text{caso contrário.} \end{cases}$

Programação Linear Inteira: formulações (cont.)

▷ Formulação:

$$
\begin{array}{rll}
\min & z = \sum_{t=1}^{n}(p_t x_t + h_t s_t + f_t y_t) & \\
\text{Sujeito a} & s_{t-1} + x_t = d_t + s_t, \quad \text{para } t = 1, \ldots, n & (1) \\
& x_t \leq C_t y_t, \quad \text{para } t = 1, \ldots, n & (2) \\
& s_t \geq 0, x_t \geq 0, \text{para } t = 1, \ldots, n, & \\
& y_t \in \{0, 1\}, \text{ para } t = 1, \ldots, n. &
\end{array}
$$

onde (1) representa a conservação de fluxo no período t e (2) restringe a produção no período t a C_t ou a *zero* dependendo se a decisão foi de produzir ou não naquele período.

Prog. Linear Inteira: *branch & bound*

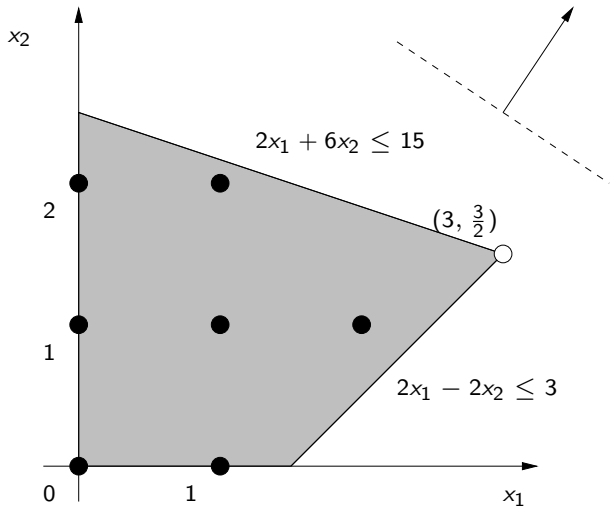
▷ *Função classificadora*: usar a relaxação linear, ou seja, resolver o problema como se todas as variáveis fossem reais.

▷ Explorar o nó com **melhor limitante** (*best bound)*.

▷ No caso de variáveis binárias, substituir $x \in \{0, 1\}$ por $0 \leq x \leq 1$.

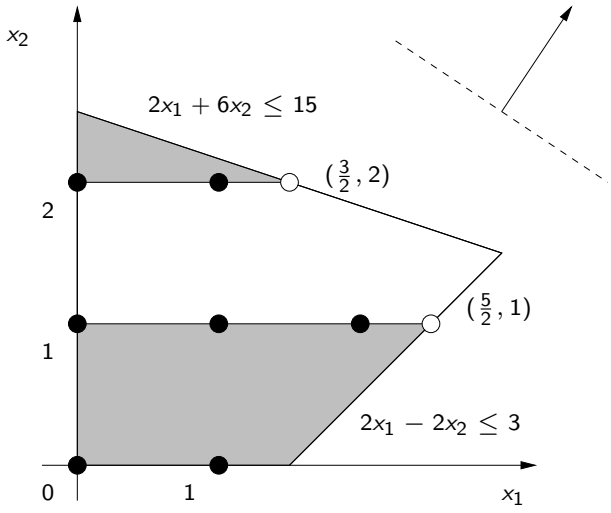
▷ *Divisão do espaço de soluções*: mais comum é usar a *regra da variável "mais fracionária"*, onde dada a solução ótima x^* da relaxação linear, encontra-se a variável x cujo máximo das diferenças $(x - \lfloor x^* \rfloor)$ e $(\lceil x^* \rceil - x)$ seja o mais próximo de 0.5 e cria-se dois PLIs a partir do PLI corrente acrescentando em um deles a restrição $x \leq \lfloor x^* \rfloor$ e no outro a restrição $x \geq \lceil x^* \rceil$.

Prog. Linear Inteira: *branch & bound* (exemplo)



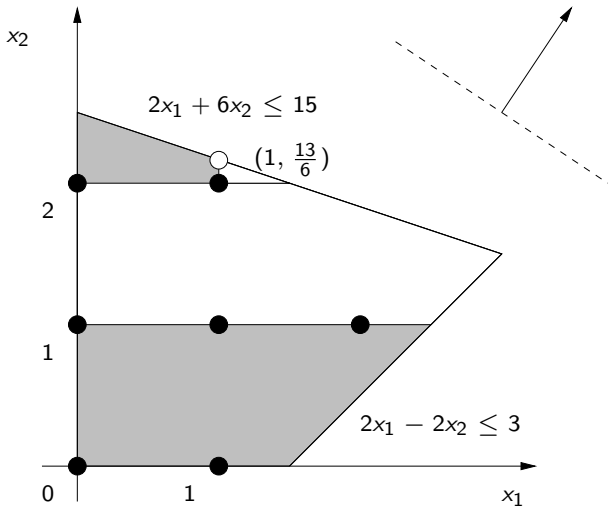
*PL*0:
$\max 2x_1 + 3x_2$
$2x_1 + 6x_2 \leq 15$
$2x_1 - 2x_2 \leq 3$
$x_1 \geq 0, x_2 \geq 0$

Prog. Linear Inteira: *branch & bound* (exemplo)



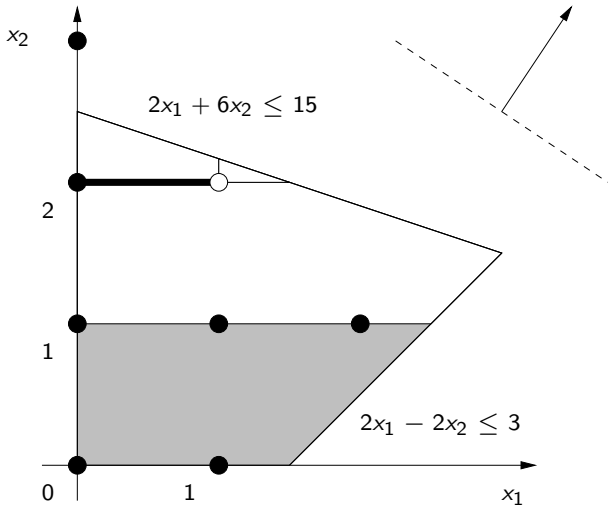
$PL1 = PL0 + \{x_2 \leq 1\}, \ (z_1 = 8)$
$PL2 = PL0 + \{x_2 \geq 2\}, \ (z_2 = 9)$

Prog. Linear Inteira: *branch & bound* (exemplo)



$PL3 = PL2 + \{x_1 \leq 1\}$, $(z_3 = 8.5)$
$PL4 = PL2 + \{x_1 \geq 2\}$, (inviável)

Prog. Linear Inteira: *branch & bound* (exemplo)



$PL5 = PL3 + \{x_2 \leq 2\}$, $(z_5 = 8)$
$PL6 = PL3 + \{x_2 \geq 3\}$, (inviável)

Prog. Linear Inteira: *branch & bound* (exemplo)