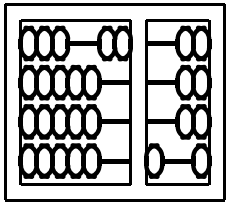


Universidade Estadual de Campinas - UNICAMP
Instituto de Computação - IC

Tratamento algorítmico de problemas $\mathcal{NP}$-difíceis

Cid Carvalho de Souza

Agosto de 2005

Objetivo

Discutir diversas técnicas de tratamento de problemas $\mathcal{NP}$-difíceis, revendo alguns paradigmas do projeto de algoritmos, usando como exemplo o problema binário da mochila (BKP).

Descrição do problema e de sua complexidade

- O problema binário da mochila (BKP):

Descrição do problema e de sua complexidade

- O problema binário da mochila (BKP):

  Entrada: um conjunto de n itens. Para cada item $i \in N = \{1, \ldots, n\}$ estão associados dois valores inteiros positivos w_i e c_i representando seu peso e seu custo, respectivamente. Dois valores inteiros positivos: W (capacidade da mochila) e C (custo alvo).

Descrição do problema e de sua complexidade

- O problema binário da mochila (BKP):

  Entrada: um conjunto de n itens. Para cada item $i \in N = \{1, \ldots, n\}$ estão associados dois valores inteiros positivos w_i e c_i representando seu peso e seu custo, respectivamente. Dois valores inteiros positivos: W (capacidade da mochila) e C (custo alvo).
  Pergunta: Existe um subconjunto I de N tal que $\sum_{i \in I} w_i \leq W$ e $\sum_{i \in I} c_i \geq C$?

Descrição do problema e de sua complexidade

- O problema binário da mochila (BKP):

  Entrada: um conjunto de n itens. Para cada item $i \in N = \{1, \ldots, n\}$ estão associados dois valores inteiros positivos w_i e c_i representando seu peso e seu custo, respectivamente. Dois valores inteiros positivos: W (capacidade da mochila) e C (custo alvo).

  Pergunta: Existe um subconjunto I de N tal que $\sum_{i \in I} w_i \leq W$ e $\sum_{i \in I} c_i \geq C$?

- Hipóteses:

Descrição do problema e de sua complexidade

- O problema binário da mochila (BKP):

  Entrada: um conjunto de n itens. Para cada item $i \in N = \{1, \ldots, n\}$ estão associados dois valores inteiros positivos w_i e c_i representando seu peso e seu custo, respectivamente. Dois valores inteiros positivos: W (capacidade da mochila) e C (custo alvo).

  Pergunta: Existe um subconjunto I de N tal que $\sum_{i \in I} w_i \leq W$ e $\sum_{i \in I} c_i \geq C$?

- Hipóteses:
  - $w_i \leq W$ para todo $i \in N$;

Descrição do problema e de sua complexidade

- O problema binário da mochila (BKP):

  Entrada: um conjunto de n itens. Para cada item $i \in N = \{1, \ldots, n\}$ estão associados dois valores inteiros positivos w_i e c_i representando seu peso e seu custo, respectivamente. Dois valores inteiros positivos: W (capacidade da mochila) e C (custo alvo).

  Pergunta: Existe um subconjunto I de N tal que $\sum_{i \in I} w_i \leq W$ e $\sum_{i \in I} c_i \geq C$?

- Hipóteses:
  - $w_i \leq W$ para todo $i \in N$;
  - $\sum_{i \in N} w_i > W$.

Descrição do problema e de sua complexidade

- O problema binário da mochila (BKP):

  Entrada: um conjunto de n itens. Para cada item $i \in N = \{1, \ldots, n\}$ estão associados dois valores inteiros positivos w_i e c_i representando seu peso e seu custo, respectivamente. Dois valores inteiros positivos: W (capacidade da mochila) e C (custo alvo).

  Pergunta: Existe um subconjunto I de N tal que $\sum_{i \in I} w_i \leq W$ e $\sum_{i \in I} c_i \geq C$?

- Hipóteses:
  - $w_i \leq W$ para todo $i \in N$;
  - $\sum_{i \in N} w_i > W$.
- Complexidade:

Descrição do problema e de sua complexidade

- O problema binário da mochila (BKP):

  Entrada: um conjunto de n itens. Para cada item $i \in N = \{1, \ldots, n\}$ estão associados dois valores inteiros positivos w_i e c_i representando seu peso e seu custo, respectivamente. Dois valores inteiros positivos: W (capacidade da mochila) e C (custo alvo).

  Pergunta: Existe um subconjunto I de N tal que $\sum_{i \in I} w_i \leq W$ e $\sum_{i \in I} c_i \geq C$?

- Hipóteses:
  - $w_i \leq W$ para todo $i \in N$;
  - $\sum_{i \in N} w_i > W$.
- Complexidade:
  - BKP está em $\mathcal{NP}$.

Descrição do problema e de sua complexidade

- O problema binário da mochila (BKP):

  Entrada: um conjunto de n itens. Para cada item $i \in N = \{1, \ldots, n\}$ estão associados dois valores inteiros positivos w_i e c_i representando seu peso e seu custo, respectivamente. Dois valores inteiros positivos: W (capacidade da mochila) e C (custo alvo).

  Pergunta: Existe um subconjunto I de N tal que $\sum_{i \in I} w_i \leq W$ e $\sum_{i \in I} c_i \geq C$?

- Hipóteses:
  - $w_i \leq W$ para todo $i \in N$;
  - $\sum_{i \in N} w_i > W$.
- Complexidade:
  - BKP está em $\mathcal{NP}$.
  - Cook (1971): SAT é $\mathcal{NP}$-completo.

Descrição do problema e de sua complexidade

- O problema binário da mochila (BKP):

  Entrada: um conjunto de n itens. Para cada item $i \in N = \{1, \ldots, n\}$ estão associados dois valores inteiros positivos w_i e c_i representando seu peso e seu custo, respectivamente. Dois valores inteiros positivos: W (capacidade da mochila) e C (custo alvo).
  Pergunta: Existe um subconjunto I de N tal que $\sum_{i \in I} w_i \leq W$ e $\sum_{i \in I} c_i \geq C$?

- Hipóteses:
  - $w_i \leq W$ para todo $i \in N$;
  - $\sum_{i \in N} w_i > W$.
- Complexidade:
  - BKP está em $\mathcal{NP}$.
  - Cook (1971): SAT é $\mathcal{NP}$-completo.
  - Karp (1972): $\texttt{SAT} \propto \texttt{CLIQUE} \propto \texttt{3DM} \propto \texttt{PART} \propto \texttt{BKP}$.

Tratamento de Problemas $\mathcal{NP}$-difíceis

- *Versão de otimização*:

Tratamento de Problemas $\mathcal{NP}$-difíceis

- *Versão de otimização*:
  - encontrar $I \subseteq N$ satisfazendo $\sum_{i \in I} w_i \leq W$ que maximiza $\sum_{i \in I} c_i$.

Tratamento de Problemas $\mathcal{NP}$-difíceis

- *Versão de otimização*:
  - encontrar $I \subseteq N$ satisfazendo $\sum_{i \in I} w_i \leq W$ que maximiza $\sum_{i \in I} c_i$.
  - A versão de otimização está em $\mathcal{NP}$-difícil: imediato.

Tratamento de Problemas $\mathcal{NP}$-difíceis

- *Versão de otimização*:
  - encontrar $I \subseteq N$ satisfazendo $\sum_{i \in I} w_i \leq W$ que maximiza $\sum_{i \in I} c_i$.
  - A versão de otimização está em $\mathcal{NP}$-difícil: imediato.
- Alternativas algorítmicas:

Tratamento de Problemas $\mathcal{NP}$-difíceis

- *Versão de otimização*:
  - encontrar $I \subseteq N$ satisfazendo $\sum_{i \in I} w_i \leq W$ que maximiza $\sum_{i \in I} c_i$.
  - A versão de otimização está em $\mathcal{NP}$-difícil: imediato.
- Alternativas algorítmicas:
  - Heurísticas: não garantem a qualidade da solução retornada.

Tratamento de Problemas $\mathcal{NP}$-difíceis

- *Versão de otimização*:
  - encontrar $I \subseteq N$ satisfazendo $\sum_{i \in I} w_i \leq W$ que maximiza $\sum_{i \in I} c_i$.
  - A versão de otimização está em $\mathcal{NP}$-difícil: imediato.
- Alternativas algorítmicas:
  - Heurísticas: não garantem a qualidade da solução retornada.
  - Exatas: fornecem uma *prova de otimalidade* para a solução retornada.

Tratamento de Problemas $\mathcal{NP}$-difíceis

- *Versão de otimização*:
  - encontrar $I \subseteq N$ satisfazendo $\sum_{i \in I} w_i \leq W$ que maximiza $\sum_{i \in I} c_i$.
  - A versão de otimização está em $\mathcal{NP}$-difícil: imediato.
- Alternativas algorítmicas:
  - Heurísticas: não garantem a qualidade da solução retornada.
  - Exatas: fornecem uma *prova de otimalidade* para a solução retornada.
  - Aproximadas: fornecem uma solução cujo custo está a uma distância máxima, absoluta ou relativa, conhecida do ótimo.

Algoritmos gulosos

- Componentes básicas de um algoritmo guloso:

Algoritmos gulosos

- Componentes básicas de um algoritmo guloso:
  - solução construída elemento a elemento;

Algoritmos gulosos

- Componentes básicas de um algoritmo guloso:
  - solução construída elemento a elemento;
  - próximo elemento a ser inserido na solução, juntamente com aqueles já presentes na solução parcial corrente deve satisfazer às restrições do problema;

Algoritmos gulosos

- Componentes básicas de um algoritmo guloso:
  - solução construída elemento a elemento;
  - próximo elemento a ser inserido na solução, juntamente com aqueles já presentes na solução parcial corrente deve satisfazer às restrições do problema;
  - próximo elemento escolhido: otimiza uma função objetivo local, não necessariamente igual à função objetivo original do problema.

Algoritmos gulosos

- Componentes básicas de um algoritmo guloso:
  - solução construída elemento a elemento;
  - próximo elemento a ser inserido na solução, juntamente com aqueles já presentes na solução parcial corrente deve satisfazer às restrições do problema;
  - próximo elemento escolhido: otimiza uma função objetivo local, não necessariamente igual à função objetivo original do problema.
  - não há *backtracking*: decisões tomadas em passos anteriores são sempre mantidas.

Algoritmos gulosos

- Componentes básicas de um algoritmo guloso:
  - solução construída elemento a elemento;
  - próximo elemento a ser inserido na solução, juntamente com aqueles já presentes na solução parcial corrente deve satisfazer às restrições do problema;
  - próximo elemento escolhido: otimiza uma função objetivo local, não necessariamente igual à função objetivo original do problema.
  - não há *backtracking*: decisões tomadas em passos anteriores são sempre mantidas.
- Uma função objetivo local para o BKP:

Algoritmos gulosos

- Componentes básicas de um algoritmo guloso:
  - solução construída elemento a elemento;
  - próximo elemento a ser inserido na solução, juntamente com aqueles já presentes na solução parcial corrente deve satisfazer às restrições do problema;
  - próximo elemento escolhido: otimiza uma função objetivo local, não necessariamente igual à função objetivo original do problema.
  - não há *backtracking*: decisões tomadas em passos anteriores são sempre mantidas.
- Uma função objetivo local para o BKP:
  - avaliar itens pela razão custo/peso;

Algoritmos gulosos

- Componentes básicas de um algoritmo guloso:
  - solução construída elemento a elemento;
  - próximo elemento a ser inserido na solução, juntamente com aqueles já presentes na solução parcial corrente deve satisfazer às restrições do problema;
  - próximo elemento escolhido: otimiza uma função objetivo local, não necessariamente igual à função objetivo original do problema.
  - não há *backtracking*: decisões tomadas em passos anteriores são sempre mantidas.
- Uma função objetivo local para o BKP:
  - avaliar itens pela razão custo/peso;
  - **idéia**: maximizar o custo por unidade de peso transportado.

Uma heurística gulosa para o BKP

```
Mochila-GULOSA(W, n, c, w);
1.  Ordenar itens em ordem decrescente da razão c_i/w_i;
2.  j ← 1;        I ← { };
3.  W' ← W;  (* capacidade residual *)
4.  C' ← 0;      (* custo da solução *)
5.  enquanto W' ≠ 0 e j ≤ n faça
6.      se w_j ≤ W' então
7.          W' ← W' − w_j;
8.          C' ← C' + c_j;
9.          I ← I ∪ {j};
10.     fim se
11.     j ← j + 1;
12. fim-enquanto
retornar (C', I);
```

Uma heurística gulosa para o BKP

- Complexidade: dominada pela ordenação dos itens na linha 1, portanto, é $O(n \log n)$.

Uma heurística gulosa para o BKP

- Complexidade: dominada pela ordenação dos itens na linha 1, portanto, é $O(n \log n)$.
- Exemplo: $n = 7$, $W = 17$

Uma heurística gulosa para o BKP

- Complexidade: dominada pela ordenação dos itens na linha 1, portanto, é $O(n \log n)$.
- Exemplo: $n = 7$, $W = 17$

item	1	2	3	4	5	6	7
custos	8	16	20	12	6	10	4
pesos	3	7	9	6	3	5	2

Uma heurística gulosa para o BKP

- Complexidade: dominada pela ordenação dos itens na linha 1, portanto, é $O(n \log n)$.
- Exemplo: $n = 7$, $W = 17$

item	1	2	3	4	5	6	7
custos	8	16	20	12	6	10	4
pesos	3	7	9	6	3	5	2
$\in I$?							
$\overline{W}$							

Uma heurística gulosa para o BKP

- Complexidade: dominada pela ordenação dos itens na linha 1, portanto, é $O(n \log n)$.
- Exemplo: $n = 7$, $W = 17$

item	1	2	3	4	5	6	7
custos	8	16	20	12	6	10	4
pesos	3	7	9	6	3	5	2
$\in I$?	S						
$\overline{W}$	14						

Uma heurística gulosa para o BKP

- Complexidade: dominada pela ordenação dos itens na linha 1, portanto, é $O(n \log n)$.
- Exemplo: $n = 7$, $W = 17$

item	1	2	3	4	5	6	7
custos	8	16	20	12	6	10	4
pesos	3	7	9	6	3	5	2
$\in I$?	S	S					
$\overline{W}$	14	7					

Uma heurística gulosa para o BKP

- Complexidade: dominada pela ordenação dos itens na linha 1, portanto, é $O(n \log n)$.
- Exemplo: $n = 7$, $W = 17$

item	1	2	3	4	5	6	7
custos	8	16	20	12	6	10	4
pesos	3	7	9	6	3	5	2
$\in I$?	S	S	N				
$\overline{W}$	14	7	7				

Uma heurística gulosa para o BKP

- Complexidade: dominada pela ordenação dos itens na linha 1, portanto, é $O(n \log n)$.
- Exemplo: $n = 7$, $W = 17$

item	1	2	3	4	5	6	7
custos	8	16	20	12	6	10	4
pesos	3	7	9	6	3	5	2
$\in I$?	S	S	N	S			
$\overline{W}$	14	7	7	1			

Uma heurística gulosa para o BKP

- Complexidade: dominada pela ordenação dos itens na linha 1, portanto, é $O(n \log n)$.
- Exemplo: $n = 7$, $W = 17$

item	1	2	3	4	5	6	7
custos	8	16	20	12	6	10	4
pesos	3	7	9	6	3	5	2
$\in I$?	S	S	N	S	N	N	N
$\overline{W}$	14	7	7	1	1	1	1

Uma heurística gulosa para o BKP

- Complexidade: dominada pela ordenação dos itens na linha 1, portanto, é $O(n \log n)$.
- Exemplo: $n = 7$, $W = 17$

item	1	2	3	4	5	6	7
custos	8	16	20	12	6	10	4
pesos	3	7	9	6	3	5	2
$\in I$?	S	S	N	S	N	N	N
$\overline{W}$	14	7	7	1	1	1	1

Solução heurística: $I = \{1, 2, 4\}$ com custo 36 (ótimo=38).

Algoritmo exato de Programação Dinâmica

- Componentes básicas de um algoritmo de Programação Dinâmica:

Algoritmo exato de Programação Dinâmica

- Componentes básicas de um algoritmo de Programação Dinâmica:
  - Uma solução ótima contém soluções ótimas de subproblemas semelhantes ao problema original;

Algoritmo exato de Programação Dinâmica

- Componentes básicas de um algoritmo de Programação Dinâmica:
  - Uma solução ótima contém soluções ótimas de subproblemas semelhantes ao problema original;
  - O valor ótimo está relacionado aos valores ótimos destes subproblemas por meio de uma fórmula de recorrência;

Algoritmo exato de Programação Dinâmica

- Componentes básicas de um algoritmo de Programação Dinâmica:
  - Uma solução ótima contém soluções ótimas de subproblemas semelhantes ao problema original;
  - O valor ótimo está relacionado aos valores ótimos destes subproblemas por meio de uma fórmula de recorrência;
  - Os valores subótimos são tabelados de modo a impedir recálculos;

Algoritmo exato de Programação Dinâmica

- Componentes básicas de um algoritmo de Programação Dinâmica:
  - Uma solução ótima contém soluções ótimas de subproblemas semelhantes ao problema original;
  - O valor ótimo está relacionado aos valores ótimos destes subproblemas por meio de uma fórmula de recorrência;
  - Os valores subótimos são tabelados de modo a impedir recálculos;
- Valores tabelados para `BKP`

Algoritmo exato de Programação Dinâmica

- Componentes básicas de um algoritmo de Programação Dinâmica:
  - Uma solução ótima contém soluções ótimas de subproblemas semelhantes ao problema original;
  - O valor ótimo está relacionado aos valores ótimos destes subproblemas por meio de uma fórmula de recorrência;
  - Os valores subótimos são tabelados de modo a impedir recálculos;
- Valores tabelados para `BKP`
  - $A[i, d]$: peso do subconjunto *mais leve* dos i primeiros itens com valor exatamente d;

Algoritmo exato de Programação Dinâmica

- Componentes básicas de um algoritmo de Programação Dinâmica:
  - Uma solução ótima contém soluções ótimas de subproblemas semelhantes ao problema original;
  - O valor ótimo está relacionado aos valores ótimos destes subproblemas por meio de uma fórmula de recorrência;
  - Os valores subótimos são tabelados de modo a impedir recálculos;
- Valores tabelados para `BKP`
  - $A[i, d]$: peso do subconjunto *mais leve* dos i primeiros itens com valor exatamente d;
  - $A[i, d] = \infty$ se não existir tal conjunto;

Algoritmo exato de Programação Dinâmica

- Componentes básicas de um algoritmo de Programação Dinâmica:
  - Uma solução ótima contém soluções ótimas de subproblemas semelhantes ao problema original;
  - O valor ótimo está relacionado aos valores ótimos destes subproblemas por meio de uma fórmula de recorrência;
  - Os valores subótimos são tabelados de modo a impedir recálculos;
- Valores tabelados para `BKP`
  - $A[i, d]$: peso do subconjunto *mais leve* dos i primeiros itens com valor exatamente d;
  - $A[i, d] = \infty$ se não existir tal conjunto;
  - **idéia**: dadas duas soluções de mesmo custo, dá-se preferência àquela que tem menos peso;

Algoritmo exato de Programação Dinâmica

- Fórmula de recorrência:

$$A[i,d] = \begin{cases} \min\{A[i-1,d], A[i-1,d-c_i] + w_i\}, & \text{se } c_i < d, \\ A[i-1,d], & \text{caso contrário.} \end{cases}$$

Algoritmo exato de Programação Dinâmica

- Fórmula de recorrência:

$$A[i,d] = \begin{cases} \min\{A[i-1,d], A[i-1,d-c_i] + w_i\}, & \text{se } c_i < d, \\ A[i-1,d], & \text{caso contrário.} \end{cases}$$

- Dimensão da matriz A: $(n+1) \times (nC+1)$.

Algoritmo exato de Programação Dinâmica

- Fórmula de recorrência:

$$A[i,d] = \begin{cases} \min\{A[i-1,d], A[i-1,d-c_i] + w_i\}, & \text{se } c_i < d, \\ A[i-1,d], & \text{caso contrário.} \end{cases}$$

- Dimensão da matriz A: $(n+1) \times (nC+1)$.
- Inicialização de A:

Algoritmo exato de Programação Dinâmica

- Fórmula de recorrência:

$$A[i,d] = \begin{cases} \min\{A[i-1,d], A[i-1,d-c_i] + w_i\}, & \text{se } c_i < d, \\ A[i-1,d], & \text{caso contrário.} \end{cases}$$

- Dimensão da matriz A: $(n+1) \times (nC+1)$.
- Inicialização de A:
  - Definição: $C = \max_{i \in N}\{c_i\}$;

Algoritmo exato de Programação Dinâmica

- Fórmula de recorrência:

$$A[i,d] = \begin{cases} \min\{A[i-1,d], A[i-1,d-c_i] + w_i\}, & \text{se } c_i < d, \\ A[i-1,d], & \text{caso contrário.} \end{cases}$$

- Dimensão da matriz A: $(n+1) \times (nC+1)$.
- Inicialização de A:
  - Definição: $C = \max_{i \in N}\{c_i\}$;
  - *Primeira coluna*: $A[i,0] = 0$, para todo $i \in N \cup \{0\}$;

Algoritmo exato de Programação Dinâmica

- Fórmula de recorrência:

$$A[i,d] = \begin{cases} \min\{A[i-1,d], A[i-1,d-c_i] + w_i\}, & \text{se } c_i < d, \\ A[i-1,d], & \text{caso contrário.} \end{cases}$$

- Dimensão da matriz A: $(n+1) \times (nC+1)$.
- Inicialização de A:
  - Definição: $C = \max_{i \in N}\{c_i\}$;
  - *Primeira coluna*: $A[i,0] = 0$, para todo $i \in N \cup \{0\}$;
  - *Primeira linha*: $A[0,d] = \infty$, para todo $d \in \{1,\ldots,nC\}$;

Algoritmo exato de Programação Dinâmica

- Fórmula de recorrência:

$$A[i,d] = \begin{cases} \min\{A[i-1,d], A[i-1,d-c_i] + w_i\}, & \text{se } c_i < d, \\ A[i-1,d], & \text{caso contrário.} \end{cases}$$

- Dimensão da matriz A: $(n+1) \times (nC+1)$.
- Inicialização de A:
  - Definição: $C = \max_{i \in N}\{c_i\}$;
  - *Primeira coluna*: $A[i,0] = 0$, para todo $i \in N \cup \{0\}$;
  - *Primeira linha*: $A[0,d] = \infty$, para todo $d \in \{1,\ldots,nC\}$;
- Valor ótimo: $C^* = \max\{d : A[n,d] \leq W\}$

(maior índice de uma coluna cujo valor da célula na última linha não excede W)

Algoritmo exato de Programação Dinâmica

- Preenchimento da matriz A:

Algoritmo exato de Programação Dinâmica

- Preenchimento da matriz A:

	0	1		$d - c_i$		d		nC
0	0							
	...							
$i-1$	0							
i	0							
	...							
n	0							

Algoritmo exato de Programação Dinâmica

- Preenchimento da matriz A:

	0	1		$d - c_i$		d		nC
0	0	∞	$\cdots$	∞	$\cdots$	∞	$\cdots$	∞
	$\cdots$							
$i-1$	0							
i	0							
	$\cdots$							
n	0							

Algoritmo exato de Programação Dinâmica

- Preenchimento da matriz A:

	0	1		$d - c_i$		d		nC
0	0	∞	$\cdots$	∞	$\cdots$	∞	$\cdots$	∞
	$\cdots$							
$i-1$	0							
i	0							
	$\cdots$							
n	0							

Algoritmo exato de Programação Dinâmica

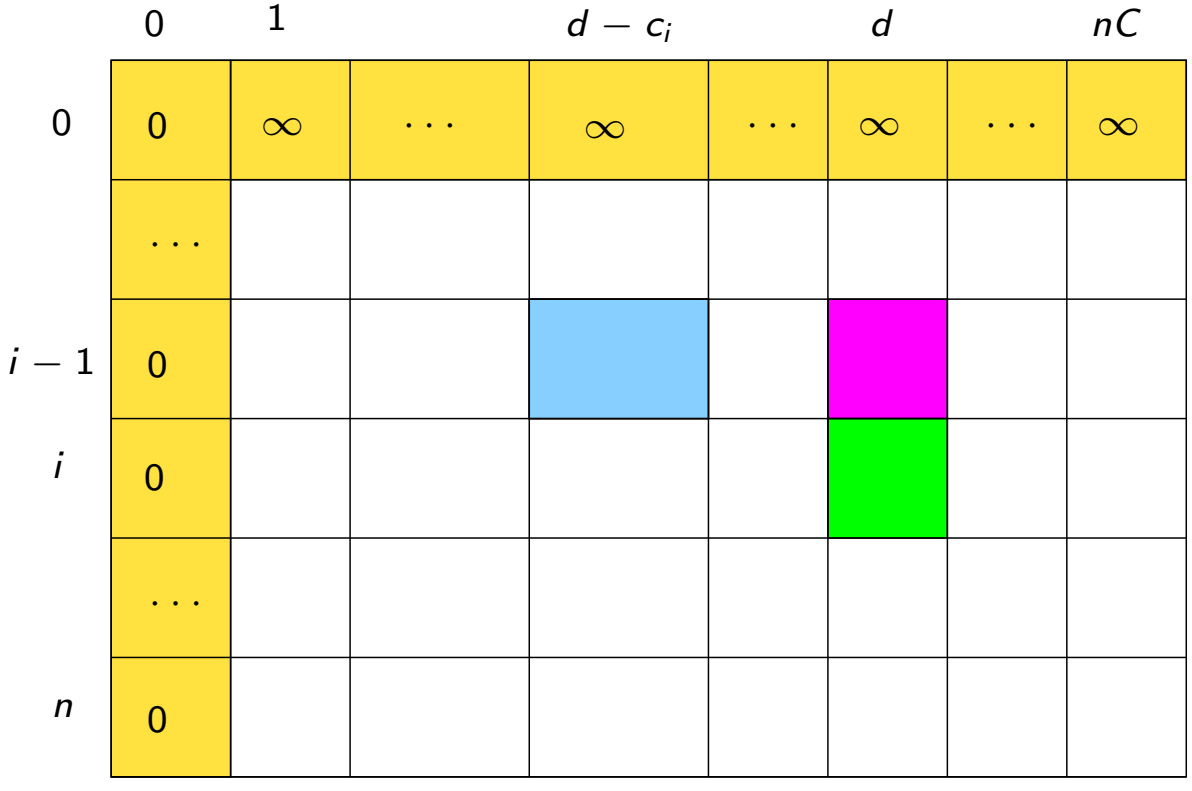
- Preenchimento da matriz A:

	0	1		$d - c_i$		d		nC
0	0	∞	$\cdots$	∞	$\cdots$	∞	$\cdots$	∞
	$\cdots$							
$i-1$	0							
i	0							
	$\cdots$							
n	0							

$$A[i,d] = \begin{cases} \min\{A[i-1,d], A[i-1,d-c_i] + w_i\}, c_i < d, \\ A[i-1,d], \text{ caso contrário.} \end{cases}$$

Algoritmo exato de Programação Dinâmica

- Preenchimento da matriz A:



$$A[i,d] = \begin{cases} \min\{A[i-1,d], A[i-1,d-c_i] + w_i\}, c_i < d, \\ A[i-1,d], \text{ caso contrário.} \end{cases}$$

Algoritmo exato de Programação Dinâmica

- Preenchimento da matriz A:

	0	1		$d - c_i$		d		nC
0	0	∞	$\cdots$	∞	$\cdots$	∞	$\cdots$	∞
	$\cdots$							
$i-1$	0							
i	0							
	$\cdots$							
n	0							

Complexidade: $O(n^2 C)$

Algoritmo exato de Programação Dinâmica

- Preenchimento da matriz A:

	0	1		$d - c_i$		d		nC
0	0	∞	$\cdots$	∞	$\cdots$	∞	$\cdots$	∞
	$\cdots$							
$i-1$	0							
i	0							
	$\cdots$							
n	0							

Complexidade: $O(n^2 C)$ *(pseudopolinomial !)*

Algoritmos aproximados

- Princípios básicos de um algoritmo aproximado:

Algoritmos aproximados

- Princípios básicos de um algoritmo aproximado:
  - Se A é um algoritmo de aproximação (relativa) para um problema de maximização então, para toda instância de entrada E, tem-se que

$$\frac{z^A(E)}{z^*(E)} \geq (1 - \epsilon),$$

onde,

Algoritmos aproximados

- Princípios básicos de um algoritmo aproximado:
  - Se A é um algoritmo de aproximação (relativa) para um problema de maximização então, para toda instância de entrada E, tem-se que

$$\frac{z^A(E)}{z^*(E)} \geq (1 - \epsilon),$$

onde,

- z^A é o valor da solução obtida por A para E;

Algoritmos aproximados

- Princípios básicos de um algoritmo aproximado:
  - Se A é um algoritmo de aproximação (relativa) para um problema de maximização então, para toda instância de entrada E, tem-se que

$$\frac{z^A(E)}{z^*(E)} \geq (1 - \epsilon),$$

onde,

- z^A é o valor da solução obtida por A para E;
- z^* é o valor ótimo para E;

Algoritmos aproximados

- Princípios básicos de um algoritmo aproximado:
  - Se A é um algoritmo de aproximação (relativa) para um problema de maximização então, para toda instância de entrada E, tem-se que

$$\frac{z^A(E)}{z^*(E)} \geq (1 - \epsilon),$$

onde,

- z^A é o valor da solução obtida por A para E;
- z^* é o valor ótimo para E;
- ϵ é uma constante no intervalo $[0, 1)$.

Algoritmos aproximados

- Princípios básicos de um algoritmo aproximado:
  - Se A é um algoritmo de aproximação (relativa) para um problema de maximização então, para toda instância de entrada E, tem-se que

$$\frac{z^A(E)}{z^*(E)} \geq (1-\epsilon),$$

    onde,
    - z^A é o valor da solução obtida por A para E;
    - z^* é o valor ótimo para E;
    - ϵ é uma constante no intervalo $[0, 1)$.
  - O valor $(1-\epsilon)$ é o fator de aproximação de A.

Um algoritmo aproximado para BKP

- Seja ϵ um valor fixo no intervalo $[0, 1)$;

Um algoritmo aproximado para BKP

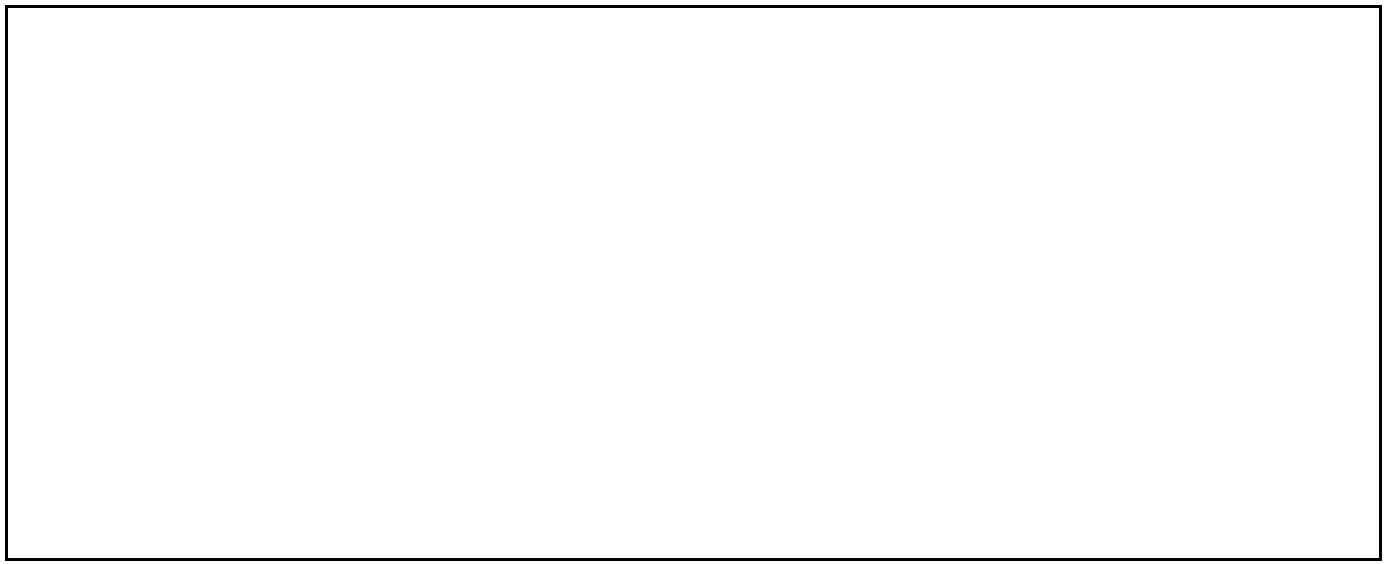
- Seja ϵ um valor fixo no intervalo $[0, 1)$;
- Seja `ProgDin-BKP` uma rotina que implementa o algoritmo exato de Programação Dinâmica;

Um algoritmo aproximado para BKP

- Seja ϵ um valor fixo no intervalo $[0, 1)$;
- Seja `ProgDin-BKP` uma rotina que implementa o algoritmo exato de Programação Dinâmica;
- Algoritmo:

Um algoritmo aproximado para BKP

- Seja ϵ um valor fixo no intervalo $[0, 1)$;
- Seja `ProgDin-BKP` uma rotina que implementa o algoritmo exato de Programação Dinâmica;
- Algoritmo:



Um algoritmo aproximado para BKP

- Seja ϵ um valor fixo no intervalo $[0, 1)$;
- Seja `ProgDin-BKP` uma rotina que implementa o algoritmo exato de Programação Dinâmica;
- Algoritmo:

`Mochila-APX`(n, W, c, w, ϵ);
1. $C \leftarrow \max_{i \in N}\{c_i\}$;
2. $K \leftarrow \frac{\epsilon C}{n}$;
3. **para** todo $i \in N$, **faça** $c'_i = \lfloor \frac{c_i}{K} \rfloor$;
4. **retorne** a solução de `ProgDin-BKP`(n, W, c', w);

fim.

- **Teorema**: `Mochila-APX` é uma $(1 - \epsilon)$ aproximação para `BKP`.

- **Teorema**: `Mochila-APX` é uma $(1-\epsilon)$ aproximação para `BKP`.

  `Prova`: E instância qualquer de `BKP`; I solução de E obtida por `Mochila-APX`; I^* uma solução ótima de E e z^* o valor ótimo.

- **Teorema**: `Mochila-APX` é uma $(1-\epsilon)$ aproximação para `BKP`.

  Prova: E instância qualquer de `BKP`; I solução de E obtida por `Mochila-APX`; I^* uma solução ótima de E e z^* o valor ótimo.

$$c'_i = \lfloor \tfrac{c_i}{K} \rfloor \Longrightarrow \begin{cases} c_i \geq K c'_i & (\star) \\ c'_i \geq (c_i/K) - 1 \text{ ou } K c'_i \geq c_i - K & (\dagger) \end{cases}$$

- **Teorema**: `Mochila-APX` é uma $(1-\epsilon)$ aproximação para `BKP`.

  Prova: E instância qualquer de `BKP`; I solução de E obtida por `Mochila-APX`; I^* uma solução ótima de E e z^* o valor ótimo.

$$c'_i = \lfloor \tfrac{c_i}{K} \rfloor \Longrightarrow \begin{cases} c_i \geq K c'_i & (\star) \\ c'_i \geq (c_i/K) - 1 \text{ ou } K c'_i \geq c_i - K & (\dagger) \end{cases}$$

$$z^A(E) = \sum_{i \in I} c_i$$

- **Teorema**: `Mochila-APX` é uma $(1-\epsilon)$ aproximação para `BKP`.

  Prova: E instância qualquer de `BKP`; I solução de E obtida por `Mochila-APX`; I^* uma solução ótima de E e z^* o valor ótimo.

$$c_i' = \lfloor \tfrac{c_i}{K} \rfloor \Longrightarrow \begin{cases} c_i \geq K c_i' & (\star) \\ c_i' \geq (c_i/K) - 1 \text{ ou } K c_i' \geq c_i - K & (\dagger) \end{cases}$$

$$z^A(E) = \sum_{i \in I} c_i \quad \geq$$

- **Teorema**: `Mochila-APX` é uma $(1-\epsilon)$ aproximação para `BKP`.

  Prova: E instância qualquer de `BKP`; I solução de E obtida por `Mochila-APX`; I^* uma solução ótima de E e z^* o valor ótimo.

$$c'_i = \lfloor \tfrac{c_i}{K} \rfloor \Longrightarrow \begin{cases} c_i \geq K c'_i & (\star) \\ c'_i \geq (c_i/K) - 1 \text{ ou } K c'_i \geq c_i - K & (\dagger) \end{cases}$$

$$z^A(E) = \sum_{i \in I} c_i \quad \geq \quad \sum_{i \in I} K c'_i \; (\star)$$

- **Teorema**: `Mochila-APX` é uma $(1-\epsilon)$ aproximação para `BKP`.

  Prova: E instância qualquer de `BKP`; I solução de E obtida por `Mochila-APX`; I^* uma solução ótima de E e z^* o valor ótimo.

$$c'_i = \lfloor \tfrac{c_i}{K} \rfloor \Longrightarrow \begin{cases} c_i \geq K c'_i & (\star) \\ c'_i \geq (c_i/K) - 1 \text{ ou } K c'_i \geq c_i - K & (\dagger) \end{cases}$$

$$\begin{aligned} z^A(E) = \textstyle\sum_{i \in I} c_i &\geq \textstyle\sum_{i \in I} K c'_i \ (\star) \\ &\geq \textstyle\sum_{i \in I^*} K c'_i \ (I \text{ ótima para custos } c') \end{aligned}$$

- **Teorema**: `Mochila-APX` é uma $(1-\epsilon)$ aproximação para `BKP`.

  Prova: E instância qualquer de `BKP`; I solução de E obtida por `Mochila-APX`; I^* uma solução ótima de E e z^* o valor ótimo.

$$c'_i = \lfloor \tfrac{c_i}{K} \rfloor \Longrightarrow \begin{cases} c_i \geq K c'_i & (\star) \\ c'_i \geq (c_i/K) - 1 \text{ ou } K c'_i \geq c_i - K & (\dagger) \end{cases}$$

$$\begin{aligned} z^A(E) = \textstyle\sum_{i \in I} c_i &\geq \textstyle\sum_{i \in I} K c'_i \ (\star) \\ &\geq \textstyle\sum_{i \in I^*} K c'_i \ (I \text{ ótima para custos } c') \\ &\geq \textstyle\sum_{i \in I^*} (c_i - K) = \sum_{i \in I^*} c_i - |I^*|K \ (\dagger) \end{aligned}$$

- **Teorema**: `Mochila-APX` é uma $(1-\epsilon)$ aproximação para `BKP`.

  Prova: E instância qualquer de `BKP`; I solução de E obtida por `Mochila-APX`; I^* uma solução ótima de E e z^* o valor ótimo.

$$c'_i = \lfloor \tfrac{c_i}{K} \rfloor \Longrightarrow \begin{cases} c_i \geq K c'_i & (\star) \\ c'_i \geq (c_i/K) - 1 \text{ ou } K c'_i \geq c_i - K & (\dagger) \end{cases}$$

$$\begin{aligned} z^A(E) = \textstyle\sum_{i \in I} c_i &\geq \textstyle\sum_{i \in I} K c'_i \ (\star) \\ &\geq \textstyle\sum_{i \in I^*} K c'_i \ (I \text{ ótima para custos } c') \\ &\geq \textstyle\sum_{i \in I^*} (c_i - K) = \sum_{i \in I^*} c_i - |I^*|K \ (\dagger) \\ &\geq \textstyle\sum_{i \in I^*} c_i - nK = \sum_{i \in I^*} c_i - \epsilon C \end{aligned}$$

- **Teorema**: `Mochila-APX` é uma $(1-\epsilon)$ aproximação para `BKP`.

  Prova: E instância qualquer de `BKP`; I solução de E obtida por `Mochila-APX`; I^* uma solução ótima de E e z^* o valor ótimo.

$$c'_i = \lfloor \tfrac{c_i}{K} \rfloor \Longrightarrow \begin{cases} c_i \geq K c'_i & (\star) \\ c'_i \geq (c_i/K) - 1 \text{ ou } K c'_i \geq c_i - K & (\dagger) \end{cases}$$

$$\begin{aligned} z^A(E) = \textstyle\sum_{i\in I} c_i &\geq \textstyle\sum_{i\in I} K c'_i \ (\star) \\ &\geq \textstyle\sum_{i\in I^*} K c'_i \ (I \text{ ótima para custos } c') \\ &\geq \textstyle\sum_{i\in I^*} (c_i - K) = \sum_{i\in I^*} c_i - |I^*|K \ (\dagger) \\ &\geq \textstyle\sum_{i\in I^*} c_i - nK = \sum_{i\in I^*} c_i - \epsilon C \\ &\geq z^* - \epsilon z^* = (1-\epsilon) z^*. \qquad \square \end{aligned}$$

Complexidade do Algoritmo

`Mochila-APX`(n, W, c, w, ϵ);

1. $C \leftarrow \max_{i \in N}\{c_i\}$;
2. $K \leftarrow \frac{\epsilon C}{n}$;
3. **para** todo $i \in N$, **faça** $c'_i = \lfloor \frac{c_i}{K} \rfloor$;
4. **retorne** a solução de `ProgDin-BKP`(n, W, c', w).

Complexidade do Algoritmo

`Mochila-APX`(n, W, c, w, ϵ);

1. $C \leftarrow \max_{i \in N}\{c_i\}$;
2. $K \leftarrow \frac{\epsilon C}{n}$;
3. **para** todo $i \in N$, **faça** $c'_i = \lfloor \frac{c_i}{K} \rfloor$;
4. **retorne** a solução de `ProgDin-BKP`(n, W, c', w).

- Dominada pela execução de `ProgDin-BKP`:
  $O(n^2 C') = O(n^2 \frac{C}{K}) = O(n^3 \frac{1}{\epsilon})$.

Complexidade do Algoritmo

`Mochila-APX`(n, W, c, w, ϵ);
1. $C \leftarrow \max_{i \in N}\{c_i\}$;
2. $K \leftarrow \frac{\epsilon C}{n}$;
3. **para** todo $i \in N$, **faça** $c'_i = \lfloor \frac{c_i}{K} \rfloor$;
4. **retorne** a solução de `ProgDin-BKP`(n, W, c', w).

- Dominada pela execução de `ProgDin-BKP`: $O(n^2 C') = O(n^2 \frac{C}{K}) = O(n^3 \frac{1}{\epsilon})$.
- `Mochila-APX` é um esquema de aproximação completamente polinomial (`FPTAS`).

Componentes básicas do *branch-and-bound*

- estratégia de divisão-e-conquista;

Componentes básicas do *branch-and-bound*

- estratégia de divisão-e-conquista;
- busca por solução é feita percorrendo-se sistematicamente a árvore de espaço de estados:

Componentes básicas do *branch-and-bound*

- estratégia de divisão-e-conquista;
- busca por solução é feita percorrendo-se sistematicamente a árvore de espaço de estados:
  - nó: uma porção do conjunto de soluções do problema;

Componentes básicas do *branch-and-bound*

- estratégia de divisão-e-conquista;
- busca por solução é feita percorrendo-se sistematicamente a árvore de espaço de estados:
  - nó: uma porção do conjunto de soluções do problema;
  - aresta: uma nova restrição adicionada ao problema original;

Componentes básicas do *branch-and-bound*

- estratégia de divisão-e-conquista;
- busca por solução é feita percorrendo-se sistematicamente a árvore de espaço de estados:
  - nó: uma porção do conjunto de soluções do problema;
  - aresta: uma nova restrição adicionada ao problema original;
- função de avaliação computa um limitante (dual) para as soluções representadas em cada nó.

Componentes básicas do *branch-and-bound*

- estratégia de divisão-e-conquista;
- busca por solução é feita percorrendo-se sistematicamente a árvore de espaço de estados:
  - nó: uma porção do conjunto de soluções do problema;
  - aresta: uma nova restrição adicionada ao problema original;
- função de avaliação computa um limitante (dual) para as soluções representadas em cada nó.
- nós da árvore são podados por um de três critérios: otimalidade, inviabilidade ou limitante.

Componentes básicas do *branch-and-bound*

- estratégia de divisão-e-conquista;
- busca por solução é feita percorrendo-se sistematicamente a árvore de espaço de estados:
  - nó: uma porção do conjunto de soluções do problema;
  - aresta: uma nova restrição adicionada ao problema original;
- função de avaliação computa um limitante (dual) para as soluções representadas em cada nó.
- nós da árvore são podados por um de três critérios: otimalidade, inviabilidade ou limitante.
- Nós não podados são ditos ativos e o algoritmo se encerra quando não há mais nós ativos.

Um algoritmo exato de *branch-and-bound* para BKP

- O sucesso de um algoritmo de *branch-and-bound* depende fortemente da qualidade dos limitantes duais !

Um algoritmo exato de *branch-and-bound* para BKP

- O sucesso de um algoritmo de *branch-and-bound* depende fortemente da qualidade dos limitantes duais !
- Cálculo de limitantes duais para o BKP:

Um algoritmo exato de *branch-and-bound* para BKP

- O sucesso de um algoritmo de *branch-and-bound* depende fortemente da qualidade dos limitantes duais !
- Cálculo de limitantes duais para o BKP:
  - Problema da Mochila Fracionária FKP:

Um algoritmo exato de *branch-and-bound* para BKP

- O sucesso de um algoritmo de *branch-and-bound* depende fortemente da qualidade dos limitantes duais !
- Cálculo de limitantes duais para o BKP:
  - Problema da Mochila Fracionária FKP:
    - Entrada: mesma do BKP

Um algoritmo exato de *branch-and-bound* para BKP

- O sucesso de um algoritmo de *branch-and-bound* depende fortemente da qualidade dos limitantes duais !
- Cálculo de limitantes duais para o BKP:
  - Problema da Mochila Fracionária FKP:
    - Entrada: mesma do BKP
    - Particularidade: itens pode ser colocados na mochila em qualquer fração.

Um algoritmo exato de *branch-and-bound* para BKP

- O sucesso de um algoritmo de *branch-and-bound* depende fortemente da qualidade dos limitantes duais !
- Cálculo de limitantes duais para o BKP:
  - Problema da Mochila Fracionária FKP:
    - Entrada: mesma do BKP
    - Particularidade: itens pode ser colocados na mochila em qualquer fração.
  - FKP é uma relaxação de BKP !

Um algoritmo exato de *branch-and-bound* para BKP

- O sucesso de um algoritmo de *branch-and-bound* depende fortemente da qualidade dos limitantes duais !
- Cálculo de limitantes duais para o BKP:
  - Problema da Mochila Fracionária FKP:
    - Entrada: mesma do BKP
    - Particularidade: itens pode ser colocados na mochila em qualquer fração.
  - FKP é uma relaxação de BKP !
  - O valor ótimo de FKP é um limite superior (dual) para o valor ótimo de BKP.

Um algoritmo exato de *branch-and-bound* para BKP

- O sucesso de um algoritmo de *branch-and-bound* depende fortemente da qualidade dos limitantes duais !
- Cálculo de limitantes duais para o BKP:
  - Problema da Mochila Fracionária FKP:
    - Entrada: mesma do BKP
    - Particularidade: itens pode ser colocados na mochila em qualquer fração.
  - FKP é uma relaxação de BKP !
  - O valor ótimo de FKP é um limite superior (dual) para o valor ótimo de BKP.
  - Nas soluções ótimas do FKP, no máximo um item é transportado em uma quantidade fracionada;

Um algoritmo exato de *branch-and-bound* para BKP

- O sucesso de um algoritmo de *branch-and-bound* depende fortemente da qualidade dos limitantes duais !
- Cálculo de limitantes duais para o BKP:
  - Problema da Mochila Fracionária FKP:
    - Entrada: mesma do BKP
    - Particularidade: itens pode ser colocados na mochila em qualquer fração.
  - FKP é uma relaxação de BKP !
  - O valor ótimo de FKP é um limite superior (dual) para o valor ótimo de BKP.
  - Nas soluções ótimas do FKP, no máximo um item é transportado em uma quantidade fracionada;
  - Algoritmos para o FKP:

Um algoritmo exato de *branch-and-bound* para BKP

- O sucesso de um algoritmo de *branch-and-bound* depende fortemente da qualidade dos limitantes duais !
- Cálculo de limitantes duais para o BKP:
  - Problema da Mochila Fracionária FKP:
    - Entrada: mesma do BKP
    - Particularidade: itens pode ser colocados na mochila em qualquer fração.
  - FKP é uma relaxação de BKP !
  - O valor ótimo de FKP é um limite superior (dual) para o valor ótimo de BKP.
  - Nas soluções ótimas do FKP, no máximo um item é transportado em uma quantidade fracionada;
  - Algoritmos para o FKP:
    1. modificação da heurística gulosa do BKP;

Um algoritmo exato de *branch-and-bound* para BKP

- O sucesso de um algoritmo de *branch-and-bound* depende fortemente da qualidade dos limitantes duais !
- Cálculo de limitantes duais para o BKP:
  - Problema da Mochila Fracionária FKP:
    - Entrada: mesma do BKP
    - Particularidade: itens pode ser colocados na mochila em qualquer fração.
  - FKP é uma relaxação de BKP !
  - O valor ótimo de FKP é um limite superior (dual) para o valor ótimo de BKP.
  - Nas soluções ótimas do FKP, no máximo um item é transportado em uma quantidade fracionada;
  - Algoritmos para o FKP:
    1. modificação da heurística gulosa do BKP;
    2. Programação Linear (PL).

Resolução exata do BKP via PLI

- Branch-and-bound para PLI: limitantes duais calculados via Programação Linear.

Resolução exata do BKP via PLI

- Branch-and-bound para PLI: limitantes duais calculados via Programação Linear.
- Modelo de PLI para o BKP:

$$\begin{array}{ll} \text{maximize} & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{Sujeito a} & w_1x_1 + w_2x_2 + \ldots + w_nx_n \leq W, \\ & x \in \mathbb{B}^n. \end{array}$$

Resolução exata do BKP via PLI

- Branch-and-bound para PLI: limitantes duais calculados via Programação Linear.
- Modelo de PLI para o BKP:

$$\begin{array}{ll} \text{maximize} & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{Sujeito a} & w_1x_1 + w_2x_2 + \ldots + w_nx_n \leq W, \\ & x \in \mathbb{B}^n. \end{array}$$

- Relaxação linear:

Resolução exata do BKP via PLI

- Branch-and-bound para PLI: limitantes duais calculados via Programação Linear.
- Modelo de PLI para o BKP:

$$\begin{array}{ll} \text{maximize} & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{Sujeito a} & w_1x_1 + w_2x_2 + \ldots + w_nx_n \leq W, \\ & x \in \mathbb{B}^n. \end{array}$$

- Relaxação linear:
  - Substituir restrições de integralidade por restrições da forma $0 \leq x_i \leq 1, \forall i \in N$.

Resolução exata do BKP via PLI

- Branch-and-bound para PLI: limitantes duais calculados via Programação Linear.
- Modelo de PLI para o BKP:

$$\begin{array}{ll} \text{maximize} & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{Sujeito a} & w_1x_1 + w_2x_2 + \ldots + w_nx_n \leq W, \\ & x \in \mathbb{B}^n. \end{array}$$

- Relaxação linear:
  - Substituir restrições de integralidade por restrições da forma $0 \leq x_i \leq 1, \forall i \in N$.
  - Ou seja, x_i = fração do item i que é colocada na mochila;

Resolução exata do BKP via PLI

- Branch-and-bound para PLI: limitantes duais calculados via Programação Linear.
- Modelo de PLI para o BKP:

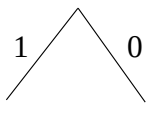
$$\begin{array}{ll} \text{maximize} & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{Sujeito a} & w_1x_1 + w_2x_2 + \ldots + w_nx_n \leq W, \\ & x \in \mathbb{B}^n. \end{array}$$

- Relaxação linear:
  - Substituir restrições de integralidade por restrições da forma $0 \leq x_i \leq 1, \forall i \in N$.
  - Ou seja, x_i = fração do item i que é colocada na mochila;
  - Logo, resolver a relaxação linear do modelo acima é equivalente a resolver o FKP !

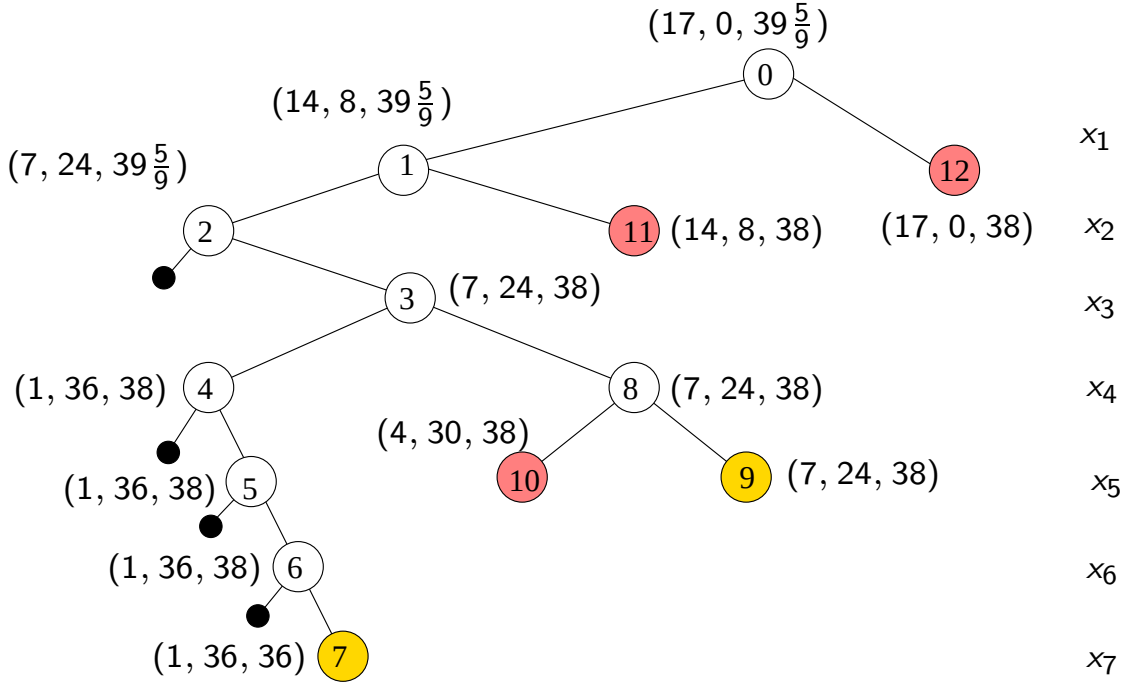
Exemplo de funcionamento do *branch-and-bound*

$n = 7$
$W = 17$

item	1	2	3	4	5	6	7
custos	8	16	20	12	6	10	4
pesos	3	7	9	6	3	5	2



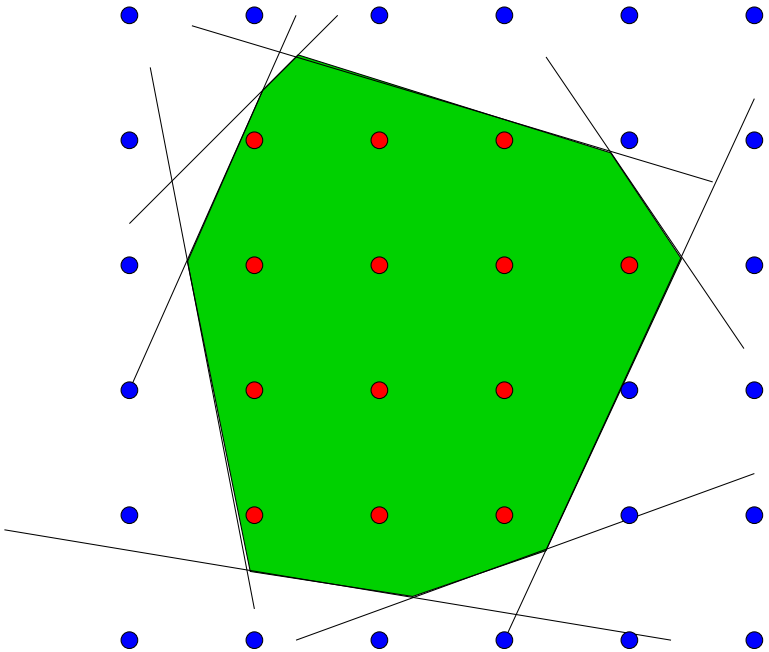
$(W', \lim_{inf}, \lim_{sup})$



Ordem de geração dos nós: 0, 1, 12, 2, 11, 3, 4, 8, 5, 6, 7, 9, 10.

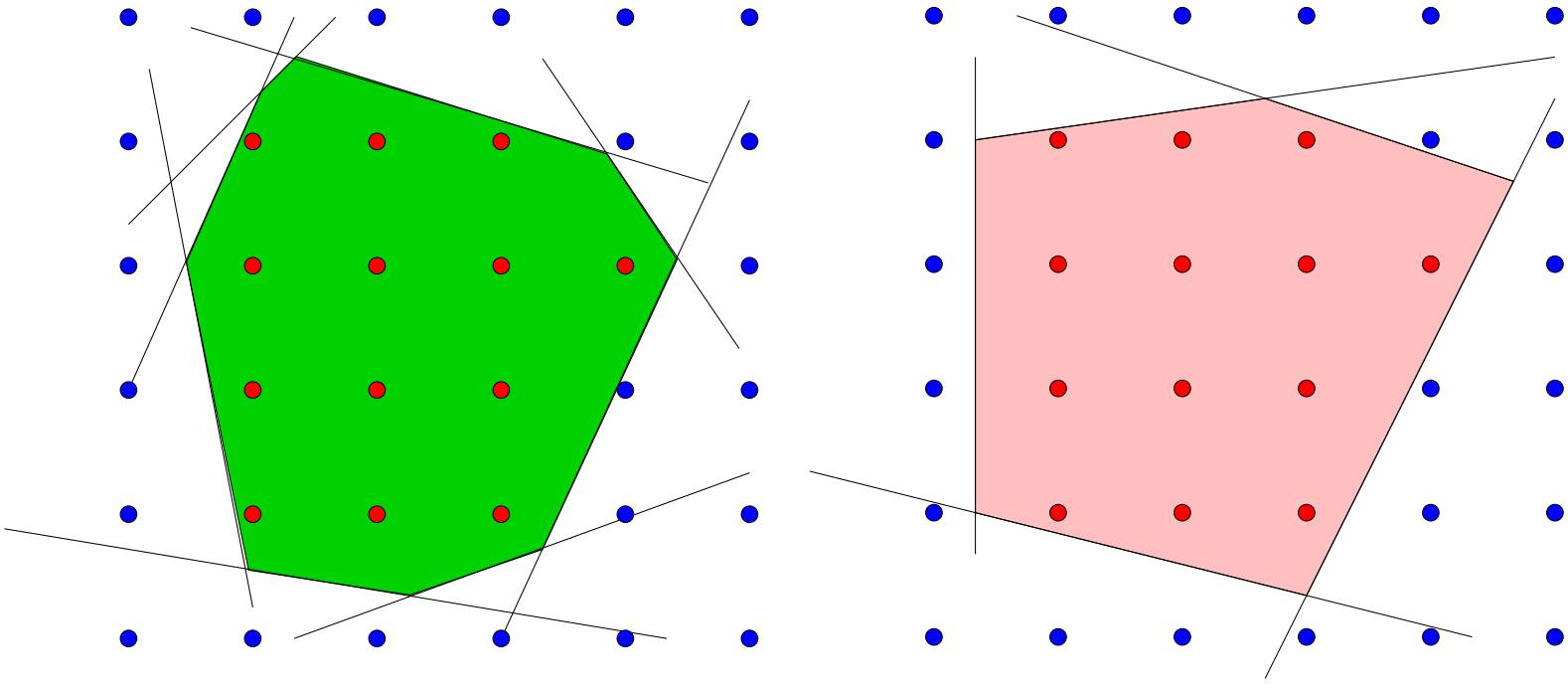
Resolução exata do BKP via PLI

- Formulação de um PLI: Como melhorar os limitantes duais ?



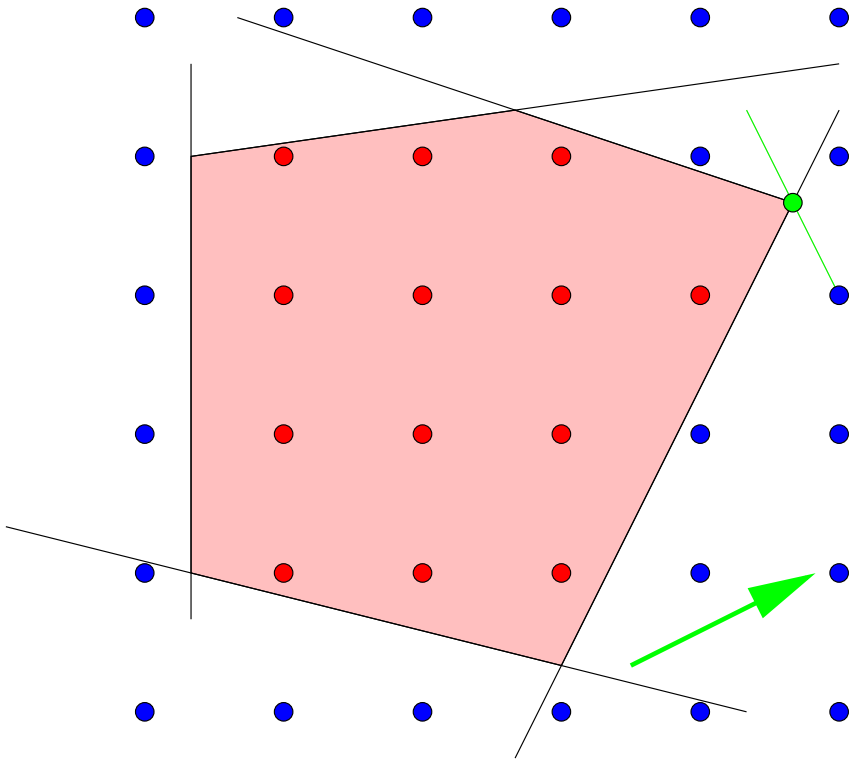
Resolução exata do BKP via PLI

- Formulações alternativas:



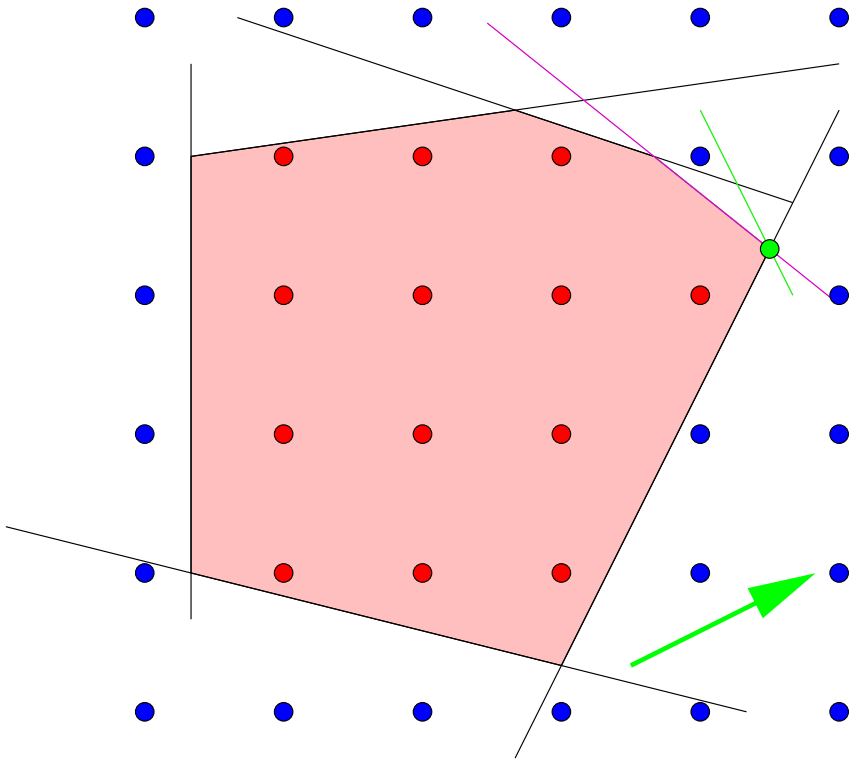
Resolução exata do BKP via PLI

- Algoritmos de planos de corte (reformulação automática do problema): problema da separação.



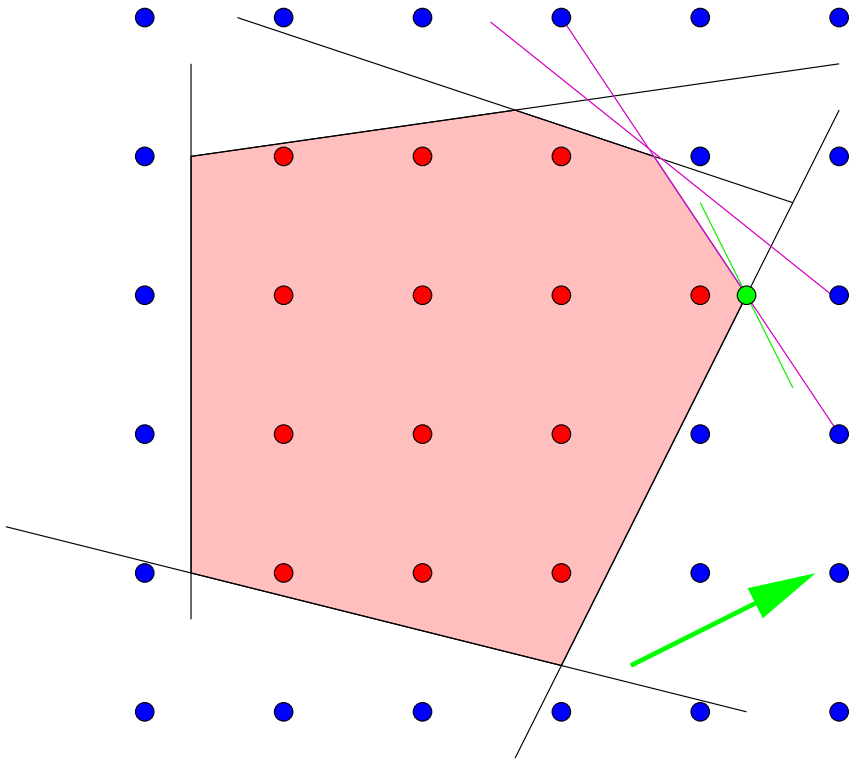
Resolução exata do BKP via PLI

- Algoritmos de planos de corte (reformulação automática do problema): problema da separação.



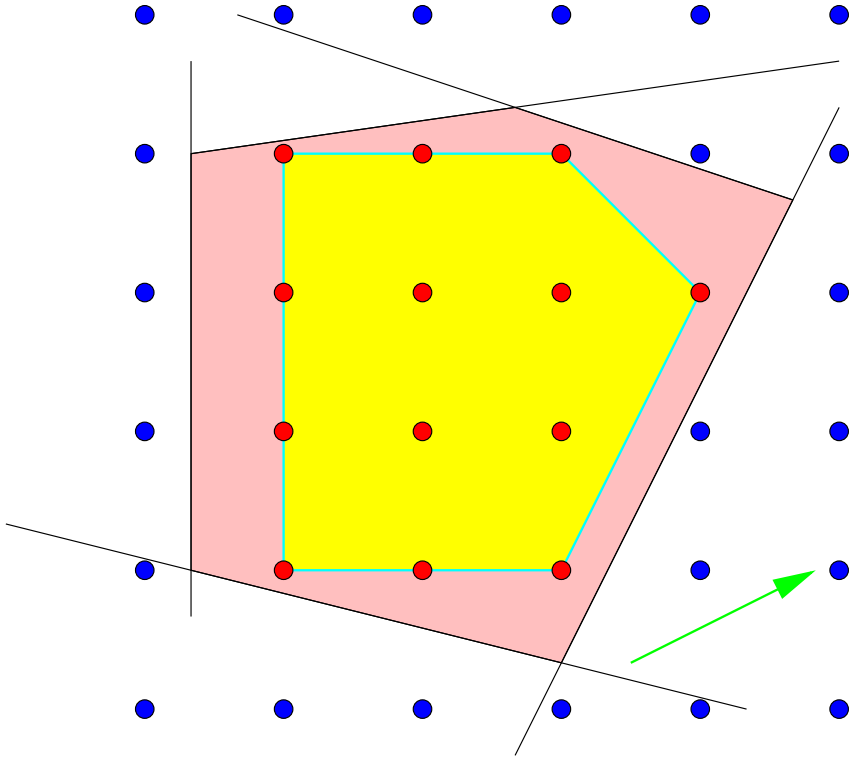
Resolução exata do BKP via PLI

- Algoritmos de planos de corte (reformulação automática do problema): problema da separação.



Resolução exata do BKP via PLI

- Envoltória Convexa das soluções inteiras: facetas



Resolução exata do BKP via PLI

- Uma desigualdade "forte" para o BKP:

Resolução exata do BKP via PLI

- Uma desigualdade "forte" para o BKP:
  - Definição: se C é um subconjunto de N tal que $\sum_{i \in C} w_i > W$, então C é uma cobertura.

Resolução exata do BKP via PLI

- Uma desigualdade "forte" para o BKP:
  - Definição: se C é um subconjunto de N tal que $\sum_{i \in C} w_i > W$, então C é uma cobertura.
  - Se C é uma cobertura, no máximo $|C| - 1$ dos seus elementos podem estar em uma solução qualquer.

Resolução exata do BKP via PLI

- Uma desigualdade "forte" para o BKP:
  - Definição: se C é um subconjunto de N tal que $\sum_{i \in C} w_i > W$, então C é uma cobertura.
  - Se C é uma cobertura, no máximo $|C| - 1$ dos seus elementos podem estar em uma solução qualquer.
  - Desigualdade linear:

$$\sum_{i \in C} x_i \leq |C| - 1.$$

Resolução exata do BKP via PLI

- Uma desigualdade "forte" para o BKP:
  - Definição: se C é um subconjunto de N tal que $\sum_{i \in C} w_i > W$, então C é uma cobertura.
  - Se C é uma cobertura, no máximo $|C| - 1$ dos seus elementos podem estar em uma solução qualquer.
  - Desigualdade linear:

$$\sum_{i \in C} x_i \leq |C| - 1.$$

  - Se a solução do FKP não for inteira, os itens com frações não-nulas formam uma cobertura.

Resolução exata do BKP via PLI

- Uma desigualdade "forte" para o BKP:
  - Definição: se C é um subconjunto de N tal que $\sum_{i \in C} w_i > W$, então C é uma cobertura.
  - Se C é uma cobertura, no máximo $|C| - 1$ dos seus elementos podem estar em uma solução qualquer.
  - Desigualdade linear:

$$\sum_{i \in C} x_i \leq |C| - 1.$$

  - Se a solução do FKP não for inteira, os itens com frações não-nulas formam uma cobertura.
  - A separação das desigualdades de cobertura (simples) é um outro problema da mochila que pode ser resolvido em tempo polinomial por Programação Dinâmica.

Combinatória Poliédrica: indo além do BKP

- Combinatória Poliédrica: estudo das desigualdades que compõem o sistema linear minimal que descreve a envoltória convexa (poliedro) das soluções inteiras de um problema.

Combinatória Poliédrica: indo além do BKP

- Combinatória Poliédrica: estudo das desigualdades que compõem o sistema linear minimal que descreve a envoltória convexa (poliedro) das soluções inteiras de um problema.
- Algoritmos de *branch-and-cut*: algoritmos de planos de corte executados de forma controlada em cada um dos nós da árvore de espaço de estados.

Combinatória Poliédrica: indo além do BKP

- Combinatória Poliédrica: estudo das desigualdades que compõem o sistema linear minimal que descreve a envoltória convexa (poliedro) das soluções inteiras de um problema.
- Algoritmos de *branch-and-cut*: algoritmos de planos de corte executados de forma controlada em cada um dos nós da árvore de espaço de estados.
- Aplicações bem-sucedidas:

Combinatória Poliédrica: indo além do BKP

- Combinatória Poliédrica: estudo das desigualdades que compõem o sistema linear minimal que descreve a envoltória convexa (poliedro) das soluções inteiras de um problema.
- Algoritmos de *branch-and-cut*: algoritmos de planos de corte executados de forma controlada em cada um dos nós da árvore de espaço de estados.
- Aplicações bem-sucedidas:
  - Crowder, Johnson e Padberg (1983): resolução de PLIs binários puros através de desigualdades do BKP (cada restrição vista como uma mochila independente).

Combinatória Poliédrica: indo além do BKP

- Combinatória Poliédrica: estudo das desigualdades que compõem o sistema linear minimal que descreve a envoltória convexa (poliedro) das soluções inteiras de um problema.
- Algoritmos de *branch-and-cut*: algoritmos de planos de corte executados de forma controlada em cada um dos nós da árvore de espaço de estados.
- Aplicações bem-sucedidas:
  - Crowder, Johnson e Padberg (1983): resolução de PLIs binários puros através de desigualdades do BKP (cada restrição vista como uma mochila independente).
  - Padberg e Rinaldi (1987, 1990, 1991): TSP.

Combinatória Poliédrica: indo além do BKP

- Combinatória Poliédrica: estudo das desigualdades que compõem o sistema linear minimal que descreve a envoltória convexa (poliedro) das soluções inteiras de um problema.
- Algoritmos de *branch-and-cut*: algoritmos de planos de corte executados de forma controlada em cada um dos nós da árvore de espaço de estados.
- Aplicações bem-sucedidas:
  - Crowder, Johnson e Padberg (1983): resolução de PLIs binários puros através de desigualdades do BKP (cada restrição vista como uma mochila independente).
  - Padberg e Rinaldi (1987, 1990, 1991): TSP.
    - www.tsp.gatech.edu/sweden/compute/compute.htm: instância com 24,978 cidades suecas;

Combinatória Poliédrica: indo além do BKP

- Combinatória Poliédrica: estudo das desigualdades que compõem o sistema linear minimal que descreve a envoltória convexa (poliedro) das soluções inteiras de um problema.
- Algoritmos de *branch-and-cut*: algoritmos de planos de corte executados de forma controlada em cada um dos nós da árvore de espaço de estados.
- Aplicações bem-sucedidas:
  - Crowder, Johnson e Padberg (1983): resolução de PLIs binários puros através de desigualdades do BKP (cada restrição vista como uma mochila independente).
  - Padberg e Rinaldi (1987, 1990, 1991): TSP.
    - www.tsp.gatech.edu/sweden/compute/compute.htm: instância com 24,978 cidades suecas;
    - *Cluster 96 PCs*, de 03/2003 a 01/2004.

Contribuições na área

- Problema do separador de vértices:

Contribuições na área

- Problema do separador de vértices:
  - *The vertex separator problem: a polyhedral investigation*, Mathematical Programming – Series A, **Julho de 2005**, vol. 103 (3), pp 583–608 (com E. Balas);

Contribuições na área

- Problema do separador de vértices:
  - *The vertex separator problem: a polyhedral investigation*, Mathematical Programming – Series A, **Julho de 2005**, vol. 103 (3), pp 583–608 (com E. Balas);
  - *The vertex separator problem: algorithms and computations*, Mathematical Programming – Series A, **Julho de 2005**, vol. 103 (3), pp 609–631 (com E. Balas).

Contribuições na área

- Problema do separador de vértices:
  - *The vertex separator problem: a polyhedral investigation*, Mathematical Programming – Series A, **Julho de 2005**, vol. 103 (3), pp 583–608 (com E. Balas);
  - *The vertex separator problem: algorithms and computations*, Mathematical Programming – Series A, **Julho de 2005**, vol. 103 (3), pp 609–631 (com E. Balas).
- Partição de Grafos:

Contribuições na área

- Problema do separador de vértices:
  - *The vertex separator problem: a polyhedral investigation,* Mathematical Programming – Series A, **Julho de 2005**, vol. 103 (3), pp 583–608 (com E. Balas);
  - *The vertex separator problem: algorithms and computations,* Mathematical Programming – Series A, **Julho de 2005**, vol. 103 (3), pp 609–631 (com E. Balas).
- Partição de Grafos:
  - *The Node Capacitated Graph Partitioning Problem: A Computational Study,* Mathematical Programming – Series B, **1998**, vol.81, pp 229–256 (com A. Martin, C. Ferreira, R. Weismantel e L. Wolsey)

Contribuições na área

- Problema do separador de vértices:
  - *The vertex separator problem: a polyhedral investigation,* Mathematical Programming – Series A, **Julho de 2005**, vol. 103 (3), pp 583–608 (com E. Balas);
  - *The vertex separator problem: algorithms and computations,* Mathematical Programming – Series A, **Julho de 2005**, vol. 103 (3), pp 609–631 (com E. Balas).
- Partição de Grafos:
  - *The Node Capacitated Graph Partitioning Problem: A Computational Study,* Mathematical Programming – Series B, **1998**, vol.81, pp 229–256 (com A. Martin, C. Ferreira, R. Weismantel e L. Wolsey)
  - *The Node Capacitated Graph Partitioning Problem: Formulations and Valid Inequalities,* Mathematical Programming – Series A, **1996**, vol.74, pp 247–266 (com A. Martin, C. Ferreira, R. Weismantel e L. Wolsey)

Considerações finais

- **Resumo**:

Considerações finais

- **Resumo**:

> Nesta apresentação discutiu-se alternativas para o tratamento de problemas $\mathcal{NP}$-difíceis, destacando-se aspectos diversos do projeto de algoritmos, além de apontar uma direção de pesquisa na área que está baseada em métodos de PLI.

Considerações finais

- **Resumo**:

> Nesta apresentação discutiu-se alternativas para o tratamento de problemas $\mathcal{NP}$-difíceis, destacando-se aspectos diversos do projeto de algoritmos, além de apontar uma direção de pesquisa na área que está baseada em métodos de PLI.

- tópicos cobertos:

Considerações finais

- **Resumo**:

> Nesta apresentação discutiu-se alternativas para o tratamento de problemas $\mathcal{NP}$-difíceis, destacando-se aspectos diversos do projeto de algoritmos, além de apontar uma direção de pesquisa na área que está baseada em métodos de PLI.

- tópicos cobertos:
  - técnicas de projeto de algoritmos: algoritmos gulosos, programação dinâmica, divisão e conquista;

Considerações finais

- **Resumo**:

> Nesta apresentação discutiu-se alternativas para o tratamento de problemas $\mathcal{NP}$-difíceis, destacando-se aspectos diversos do projeto de algoritmos, além de apontar uma direção de pesquisa na área que está baseada em métodos de PLI.

- tópicos cobertos:
  - técnicas de projeto de algoritmos: algoritmos gulosos, programação dinâmica, divisão e conquista;
  - tratamento de problemas $\mathcal{NP}$-difíceis: heurísticas, algoritmos aproximados, métodos exatos;

Considerações finais

- **Resumo**:

> Nesta apresentação discutiu-se alternativas para o tratamento de problemas $\mathcal{NP}$-difíceis, destacando-se aspectos diversos do projeto de algoritmos, além de apontar uma direção de pesquisa na área que está baseada em métodos de PLI.

- tópicos cobertos:
  - técnicas de projeto de algoritmos: algoritmos gulosos, programação dinâmica, divisão e conquista;
  - tratamento de problemas $\mathcal{NP}$-difíceis: heurísticas, algoritmos aproximados, métodos exatos;
  - algoritmos polinomiais e pseudo-polinomiais.

Considerações finais

- **Resumo**:

> Nesta apresentação discutiu-se alternativas para o tratamento de problemas $\mathcal{NP}$-difíceis, destacando-se aspectos diversos do projeto de algoritmos, além de apontar uma direção de pesquisa na área que está baseada em métodos de PLI.

- tópicos cobertos:
  - técnicas de projeto de algoritmos: algoritmos gulosos, programação dinâmica, divisão e conquista;
  - tratamento de problemas $\mathcal{NP}$-difíceis: heurísticas, algoritmos aproximados, métodos exatos;
  - algoritmos polinomiais e pseudo-polinomiais.

FIM !