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

- Aplicado a problemas onde se quer otimizar uma *função objetivo*.
- 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 $j = 1$ **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)

```
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
```

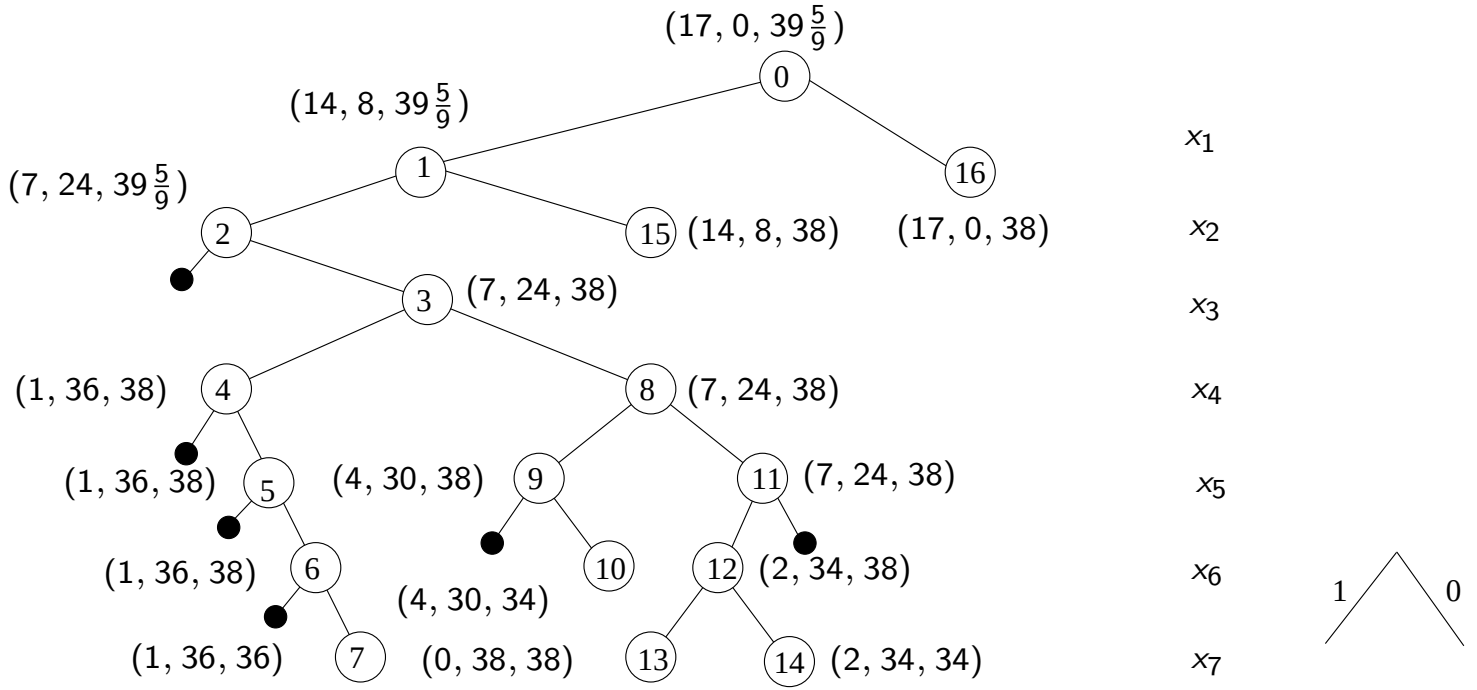
Branch&bound: mochila binária (BKP)

- 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.

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 cada uma delas composta de duas subtarefas sendo que a primeira deve ser executada na máquina 1 e a segunda na máquina 2, mas somente após encerrada a execução da primeira. A duração do processamento da tarefa j na máquina i é dado por d_{ij}.
- ▷ **Definição**: o *tempo de término* da tarefa j na máquina i é dado por f_{ij}.
- ▷ **Pede-se**: encontrar uma seqüencia de execução das subtarefas nas máquinas de modo a *minimizar a soma dos tempos de término na máquina 2*. 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 seqüencia de execução das tarefas é a mesma nas duas máquinas (**permutation schedules**) e no qual não há tempo ocioso *desnecessário* entre as tarefas.
- ▷ Exemplo: $n = 3$.

d_{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
Tempo			3				7	8

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

Máquina 1	2	2	2	3	3	1	1		
Máquina 2				2		3	3	3	1
Tempo				4				8	9

Branch&bound: FSP (cont.)

- Representação da solução: como existe uma soluçao ótima que é um *permutation schedule*, o natural é 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:

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

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

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

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

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

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

Branch&bound: FSP (cont.)

- ▷ A minimização de S_1 pode ser obtida ordenando-se as tarefas na ordem crescente dos valores de d_{1,t_k}.
- ▷ A minimização de S_2 pode ser obtida ordenando-se as tarefas na ordem crescente dos valores de d_{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.

Branch&bound: FSP (cont.)