Algoritmos

Pedro Hokama

Fontes

- [*clrs*] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
- [*timr*] Algorithms Illuminated Series, Tim Roughgarden
- Desmistificando Algoritmos, Thomas H. Cormen.

Apresentação Baseada:

- Stanford Algorithms
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V`
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZl7A5HEK6`
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420

Qualquer erro é de minha responsabilidade.

Limitantes

Em um problema de maximização:

- Um limitante superior é um valor maior ou igual que o ótimo. Pode ser obtido através de alguma relaxação do problema por exemplo. (limitante dual)
- Um limitante inferior é um valor menor ou igual que o ótimo. Poder ser obtido por uma heurística por exemplo. (limitante primal)

Exemplo: Problema da Mochila Binária.

Itens	1	2	3	4	5
Pesos	2	5	4	6	5
Valor	10	11	10	13	14

Capacidade da Mochila: 14

Limitante Inferior:

- Qualquer solução pode ser considerada um limitante inferior.
- Por exemplo se pegarmos os itens na ordem que foram dados até encher a mochila.

Pesos	**2**	**5**	**4**	6	5
Valor	**10**	**11**	**10**	13	14

Solução: 31

- A solução ótima que tem valor z^* é pelo menos 31. Ou seja $z^* \geq 31$
- Podemos tentar fortalecer esse limitante.
- E se escolhermos os itens na ordem dos que tem o melhor custo-benefício (valor/peso)

Valor/Peso (Custo-Benefício) 5 2.2 2.5 2.166 2.8

Pesos	**2**	5	**4**	6	**5**
Valor	**10**	11	**10**	13	**14**

Solução: 34, ou seja $z^* \geq 34$

Limitante Superior (Dual)

Itens	1	2	3	4	5
Pesos	2	5	4	6	5kg
Valor	10	11	10	13	14

- Se conseguíssemos preencher toda a mochila com o item que vale mais a cada kg?

Valor/Peso (Custo-Benefício) 5 2.2 2.5 2.166 2.8

- Se enchermos os 14kg que cabem na mochila com 5$ por kg.
- Um limitante superior para esse problema é $14 * 5 = 70$, ou seja $z^* \leq 70$
- Ou seja é verdade que nenhuma solução (incluindo a ótima) pode ser maior que 70.
- Esse limitante claro é bem fraco. E se enchermos a mochila só com os itens que temos até encher completamente a mochila, relaxando a integralidade.

Itens	1	2	3	4	5
Pesos	2	5	4	6	5kg
Valor	10	11	10	13	14

Valor/Peso (Custo-Benefício) 5 2.2 2.5 2.166 2.8

Pegamos o item 1, depois pegamos o item 5, depois o item 3 e enchemos o resto da mochila com 0.6 do item 2.

$$10 + 0.6 * 11 + 10 + 14 = 40.6$$

- Concluímos que a solução ótima pode ser no máximo 40.6.
- De fato como todos os valores são inteiros, podemos afirmar que $z^* \leq 40$

Itens	1	2	3	4	5
Pesos	2	5	4	6	5kg
Valor	10	11	10	13	14

- Concluimos que $34 \leq z^* \leq 40$.
- Se encontrarmos uma solução com custo igual ao limitante superior (40 no exemplo), temos certeza que ela é ótima.
- Mas a solução ótima para esse problema é a seguinte

Pesos	**2**	5	4	**6**	**5kg**
Valor	**10**	11	10	**13**	**14**

Solução: 37.

Problema de Virtualização de Máquinas

São dados um conjunto de tarefas S, cada tarefa $s \in S$ com um tempo de processamento t_s, e um tempo limite C. Cada tarefa precisa ser processada em apenas uma máquina, mas cada máquina processa várias tarefas (somando seus tempos), todas as tarefas precisam ser processadas até C.
Desejamos minimizar a quantidade necessária de máquinas virtuais.

Branch-and-Bound

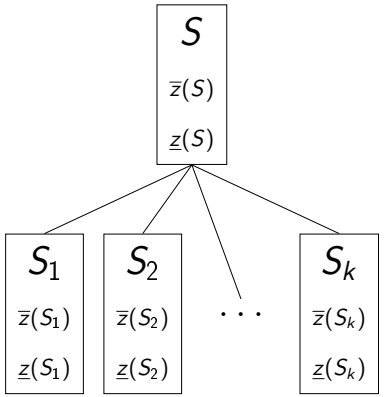
- Técnica empregada na resolução de problemas difíceis de otimização combinatória.
- O BnB enumera implicitamente todas as soluções do problema.
- Implicitamente, pois se explorasse de fato todas as soluções seria equivalente a um algoritmo de força-bruta.

- Os algoritmos de BnB são uma aplicação do paradigma de divisão e conquista.
- O Branch-and-Bound tem duas partes principais, que como você pode imaginar são:
  - *Branch*: ramificação, que é o processo de dividir o problema.
  - *Bound*: que é a busca por limitantes (inferiores e superiores) para cada subproblema.

Branch-and-Bound

- Considere um problema de maximização qualquer, em que desejamos encontrar a solução ótima.
- Seja S o conjunto de todas as soluções viáveis.
- Seja $f : S \to \mathbb{R}$ a função objetivo que mapeia cada solução $s \in S$ a um valor real.

- Vamos considerar que podemos calcular bons limitantes superiores e inferiores para um conjunto $S' \subseteq S$.
  - Seja $\overline{z}(S')$ um limitante dual (superior) para S', ou seja, um valor tal que nenhuma solução de S' tenha um valor de função objetivo maior que $\overline{z}(S')$. $f(s \in S') \leq \overline{z}(S')$
  - Seja $\underline{z}(S')$ um limitante primal (inferior) para S', ou seja, existe (com certeza) alguma solução em S' que tenha valor igual ou superior a $\underline{z}(S')$. $max\{f(s)|s \in S'\} \geq \underline{z}(S')$
- Seja $\{S_1, S_2, \ldots, S_k\}$ uma partição de S, ou seja, $S_1 \cup S_2 \cup \ldots \cup S_k = S$.
- Podemos calcular os limitantes primais e duais tanto para S quanto para S_1, S_2, ..., S_k.

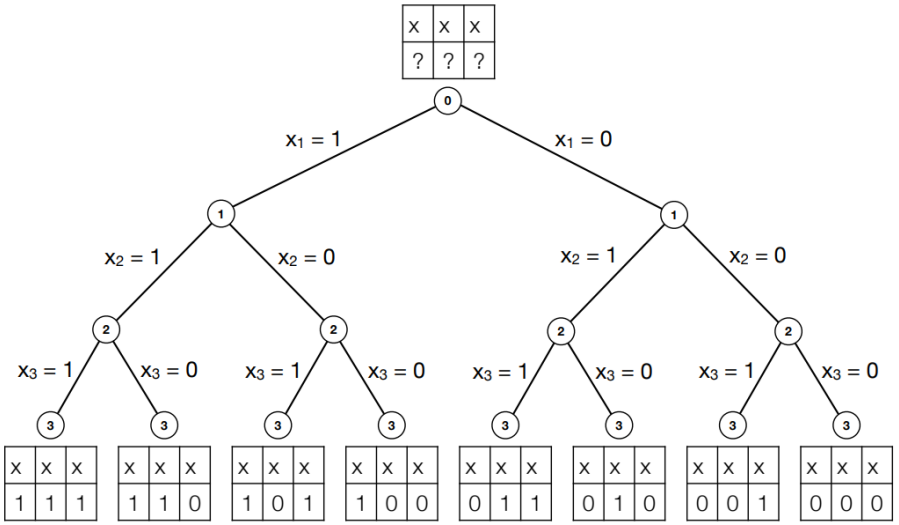


- Suponha que $\overline{z}(S_i) \leq \underline{z}(S_j)$. Como estamos buscando a solução ótima, ou seja, a solução com o maior valor. Sabemos que a melhor solução em S_i vai ser pior (ou igual) a melhor solução em S_j e por isso podemos buscar a solução apenas em S_j, *podando* S_i. Essa é a comumente chamada **poda por limitante**.

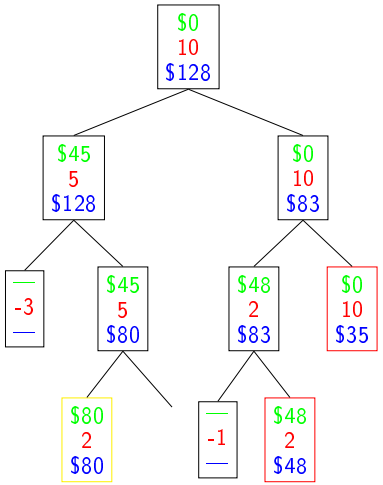
$$\begin{aligned} \text{maximize} \quad & 45x_1 + 48x_2 + 35x_3 \\ \text{subject to} \quad & \\ & 5x_1 + 8x_2 + 3x_3 \leq 10 \\ & x_i \in \{0, 1\} \quad (i \in 1..3) \ [1] \end{aligned}$$

- Como vamos subdividir o espaço de busca? Vamos dividir entre as decisões que podemos fazer em cada item.

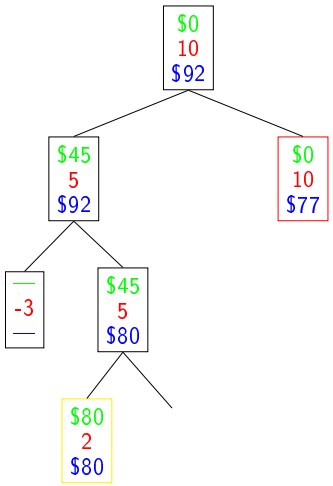
[1] Discrete Optimization, Professor Pascal Van Hentenryck



- Vamos simular a exploração
- Limitante inferior: Os itens que já estão na solução parcial.
- Limitante superior: Vamos relaxar a capacidade da mochila. (Ainda vamos verificar quando a mochila ultrapassa a capacidade)
- Vamos denotar cada solução pelo valor do limitante inferior (verde) a capacidade residual (vermelho) e o limitante superior (azul).



- Limitante superior: Vamos relaxar a integralidade da mochila, ou seja, podemos pegar uma fração de um item.

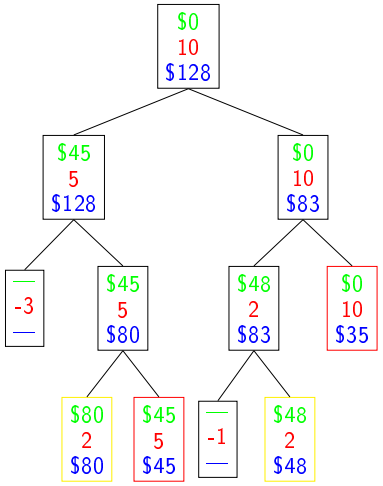


Branching Strategies

Estratégias de Ramificação

- Podemos elaborar diferentes estratégias para explorar os nós de uma árvore de branch-and-bound.
- A Depth First (que vimos anteriormente) sempre desce o mais profundo possível (indo sempre o mais a esquerda possível e depois para a direita)
- Cada estratégia pode ser um desempenho diferente dependendo do problema (e da instância).
- Mas podemos analisar alguns aspectos. Por exemplo, qual a eficiência de memória do Depth First?

Best-First



Best-First

- Sempre expande o nó "Mais promissor", ou seja, aquele que tem o maior limitante superior (no caso de um problema de maximização)
- Ele poda sempre que encontra um nó em que o limitante já é pior do que a melhor solução conhecida.
- Ele é eficiente em memória? Não muito, pode ter um número muito grande de nós ativos e cada nó precisa conter uma forma de recuperar a solução parcial.
- Ele é fácil de implementar? Normalmente é mais difícil do que o Depth-First

Ramificações

- A ideia então é dividir o espaço de soluções, calculando os limitantes e podando os nós que não fornecem soluções promissoras.
- A forma de dividir, depende do problema e da estratégia aplicada. A ideia é que a cada divisão represente uma escolha diferente.

- Por exemplo, no problema da mochila uma divisão pode ser decisão por colocar o item i na mochila e a decisão de não colocar o item i na mochila.
- Nesse exemplo obteríamos uma árvore binária, e cada nível dessa árvore pode representar a escolha de um item diferente.

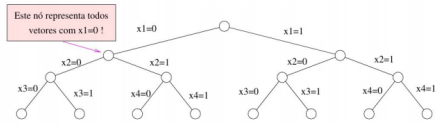


Figura 1.8. Árvore de enumeração completa para o problema binário da mochila com 3 itens.

[2]

[2] https://ic.unicamp.br/ fkm/lectures/intro-otimizacao.pdf

Outros exemplos:

- No problema do caixeiro viajante cada escolha poderia ser a escolha do próximo vértice a visitar, e nesse caso a árvore não seria binária.
- No problema de encontrar a clique máxima, uma escolha poderia ser a inclusão de um item na clique.
- No problema do *bin packing*, uma escolha poderia ser uma atribuição de um item a um contêiner.
- etc.

Algoritmo[3]

1. `B&B`; (* considerando problema de **maximização** *)
2. `Ativos` $\leftarrow$ {nó raiz}; `melhor-solução` $\leftarrow$ {}; $\underline{z} \leftarrow -\infty$;
3. **Enquanto** (`Ativos` não está vazia) **faça**
4. Escolher um nó k em `Ativos` para ramificar;
5. Remover k de `Ativos`;
6. Gerar os filhos de k: $n_1, \ldots, n_q$ computando $\overline{z}_{n_i}$ e $\underline{z}_{n_i}$ correspondentes;
   (* definir $\overline{z}_{n_i}$ e $\underline{z}_{n_i}$ iguais a $-\infty$ para subproblemas inviáveis *)
7. **Para** $j = 1$ **até** q **faça**
8. **se** ($\overline{z}_{n_i} \leq \underline{z}$) **então** podar o nó n_i; (* inclui os 3 casos *)
9. **se não**
10. **Se** (n_i representa uma única solução) **então** (* atualizar melhor limitante primal *)
11. $\underline{z} \leftarrow \overline{z}_{n_i}$; `melhor-solução` $\leftarrow$ {solução de n_i};
12. **se não** adicionar n_i à lista `Ativos`.

[3] https://ic.unicamp.br/ fkm/lectures/intro-otimizacao.pdf

Outros ramificações:

- Least-Discrenpancy: Ramifica em ondas, a cada onda ele permite que uma curva a mais aconteça na árvore.