CI1238 - Otimização

Aula 13 - *Branch-and-Bound*

Professor Murilo V. G. da Silva

Departamento de Informática

Universidade Federal do Paraná

2026 / Primeiro Semestre

Limitantes - *Branch-and-Bound*

Backtracking: Podamos da árvore os ramos com **soluções inviáveis**.

- Que outros ramos podemos podar?
  - Ramos que garantidamente **não tem solução ótima**.
- Considere a melhor solução encontrada pelo algoritmo até o momento,
  - a estratégia é descartar ramos que garantidamente não tem (nem mesmo) solução melhor do que esta solução já encontrada.
- Livro para este tópico: [KS99]

Limitantes - Branch-and-Bound

Formalizando:

- Dado problema de maximização com função objetivo $p()$ e solução ótima OPT.
- Seja $X = [x_0, ..., x_{l-1}]$ uma solução viável **parcial**

  Seja $P(X)$ o maior valor que $p()$ pode atinge na subárvore de X, i.e.,

  - $P(X) = \max\{p(X') \mid X' \text{ é solução viável na árvore enraizada em } X\}$

    $= \max\{p(X') \mid X' \text{ é solução viável extendida a partir de } X\}$
  - Note: Se $X = [\,]$, então $P(X) = OPT$
- Em geral, computar $P()$ é custoso
- Ideia chave: Encontrar função $B()$ que não seja custosa de computar tal que:
  - Para toda solução parcial viável X, $P(X) \leq B(X)$.
    i.e., computar um limitante superior para soluções na árvore de X
  - Se obtemos $B(X) \leq$ OPT_P, onde OPT_P é a melhor solução encontrada até o momento, podamos a árvore de X, pois $P(X) \leq B(X) \leq$ OPT_P

Voltando ao problema da MOCHILA

Algumas considerações a respeito do problema MOCHILA:

- Podemos permutar os itens da instância e ainda temos a "mesma" instância
- Idéia de algoritmo: Ordene os itens por valor e use um algoritmo guloso
  - **Exercício**: Mostre que o algoritmo guloso acima pode falhar
- Idéia de algoritmo: Ordene os itens por valor/peso e use um algoritmo guloso
  - **Exercício**: Mostre que este algoritmo guloso também pode falhar
- Embora o segundo algoritmo falhe, ele será útil:
  - O algoritmo guloso resolve uma versão mais simples do problema, conhecido como MOCHILA FRACIONÁRIO
  - Vamos usar isso em um algortimo *BB* para MOCHILA

Mochila Fracionário

MOCHILA FRACIONÁRIO

Instância: (p, w, n, M), onde p é um vetor de n valores reais, w é um vetor de n pesos reais e M um número real.

Resposta: lista $[x_0, ..., x_{n-1}]$ tal que $x_i \in \mathbb{R}$ e $0 \leq x_i \leq 1$, sujeito a $\sum_{i=0}^{n-1} x_i w_i \leq M$ e $\sum_{i=0}^{n-1} x_i v_i$ seja máximo.

Obs: Se modelarmos MOCHILA com PL, MOCHILA FRACIONÁRIO corresponde ao PLR

$\rightarrow$ Ou seja, sabemos que MOCHILA FRACIONÁRIO pode ser resolvido eficentemente.

$\rightarrow$ Mas não vamos usar PL, pois um algoritmo guloso já resolve o problema.

Algoritmo Guloso para Mochila Fracionário

MOCHILA FRACIONÁRIO

Instância: (p, w, n, M), onde p é um vetor de n valores reais, w é um vetor de n pesos reais e M um número real.

Resposta: lista $[x_0, ..., x_{n-1}]$ tal que $x_i \in \mathbb{R}$ e $0 \leq x_i \leq 1$, sujeito a $\sum_{i=0}^{n-1} x_i w_i \leq M$ e $\sum_{i=0}^{n-1} x_i v_i$ seja máximo.

Algorithm 4.8: RKNAP $(p_0, p_1, \ldots, p_{n-1}, w_0, w_1, \ldots, w_{n-1}, M)$

permute the indices so that $p_0/w_0 \geq p_1/w_1 \geq p_{n-1}/w_{n-1}$
$i \leftarrow 0$
$P \leftarrow 0$
$W \leftarrow 0$
for $j \leftarrow 0$ **to** $n-1$
  do $x_j \leftarrow 0$
while $W < M$ **and** $i < n$
  do **if** $W + w_i \leq M$
    then $x_i \leftarrow 1$
      $W \leftarrow W + w_i$
      $P \leftarrow P + p_i$
      $i \leftarrow i + 1$
    else $x_i \leftarrow (M - W)/w_i$
      $W \leftarrow M$
      $P \leftarrow P + x_i p_i$
      $i \leftarrow i + 1$
return (P)

BB aplicado ao problema da Mochila

Dada uma solução viável parcial $X = [x_0, ..., x_{l-1}]$, vamos definir $B(X)$ assim:

$$B(X) = \sum_{i=0}^{\ell-1} p_i x_i + \mathrm{RKNAP}(p_\ell, \ldots, p_n, w_\ell, \ldots, w_n, M - \sum_{i=0}^{\ell-1} w_i x_i)$$

$$= \sum_{i=0}^{\ell-1} p_i x_i + \mathrm{RKNAP}(p_\ell, \ldots, p_n, w_\ell, \ldots, w_n, M - \mathrm{CurW})$$

Portanto, $B(X)$ é a soma

- (1) dos valores obtidos pelos itens $0, ..., l-1$
- (2) mais o valor da solução para os itens restantes na *versão fracionária do problema* para o espaço que sobrou na mochila, que vamos chamar de CurW.
- Observe o seguinte:
  - Se as variáveis x_i fossem binárias em (2), teríamos $P(X) = B(X)$
  - Como $x_i \in \mathbb{R}$, a estimativa é mais frouxa, i.e., $P(X) \leq B(x)$, mas computável eficientemente.

BB aplicado ao problema da Mochila

Suponha que temos uma instância de MOCHILA com os itens ordenados por $\frac{p_i}{w_i} \leq \frac{p_{i+1}}{w_{i+1}}$.

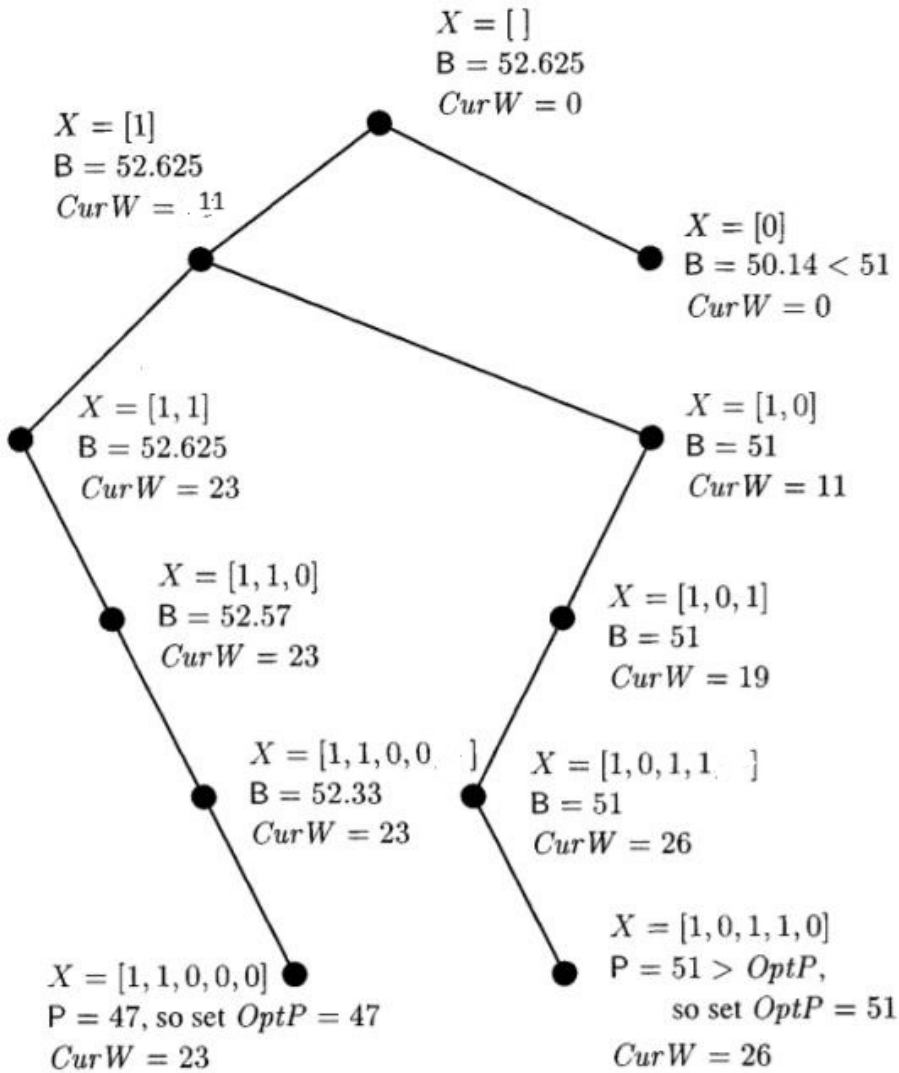
$\rightarrow$ isso pode ser feito em um pré-processamento facilmente

Algorithm 4.9: KNAPSACK3 $(\ell, CurW)$

external RKNAP()
global $X, OptX, OptP, C_\ell \quad (\ell = 0, 1, \ldots)$
if $\ell = n$
then { **if** $\sum_{i=0}^{n-1} p_i x_i > OptP$
  then { $OptP \leftarrow \sum_{i=0}^{n-1} p_i x_i$
    $OptX \leftarrow [x_0, \ldots, x_{n-1}]$
if $\ell = n$
then $C_\ell \leftarrow \emptyset$
else { **if** $CurW + w_\ell \leq M$
  then $C_\ell \leftarrow \{1, 0\}$
  else $C_\ell \leftarrow \{0\}$
$B \leftarrow \sum_{i=0}^{\ell-1} p_i x_i + \text{RKNAP}(p_\ell, \ldots, p_n, w_\ell, \ldots, w_n, M - CurW)$
for each $x \in C_\ell$
do { **if** $B \leq OptP$ **then return**
  $x_\ell \leftarrow x$
  KNAPSACK3$(\ell + 1, CurW + w_\ell x_\ell)$

Obs: Note que, como a instância já vem ordenada, podemos remover a primeira instrução dentro de RKNAP (*"permute the indices..."*)

BB aplicado ao problema da Mochila



Execução para $v = [23, 24, 15, 13, 16]$, $w = [11, 12, 8, 7, 9]$ e $M = 26$

BB aplicado ao problema da Mochila

n	Algorithm 4.1	Algorithm 4.3	Algorithm 4.9
8	511	332	52
8	511	312	78
8	511	333	72
8	511	321	74
8	511	313	57
12	8191	4598	109
12	8191	4737	93
12	8191	5079	164
12	8191	4988	195
12	8191	4620	87
16	131071	73639	192
16	131071	72302	58
16	131071	76512	168
16	131071	78716	601
16	131071	78510	392
20	2097151	1173522	299
20	2097151	1164523	104
20	2097151	1257745	416
20	2097151	1152046	118
20	2097151	1166086	480
24	33554431	19491410	693
24	33554431	18953093	180
24	33554431	17853054	278
24	33554431	19814875	559
24	33554431	18705548	755

Exemplos com instâncias aleatórias

Algoritmo de Branch-and-Bound genérico

Algorithm 4.7: BOUNDING (ℓ)

external P(), B()
global $X, OptP, OptX, \mathcal{C}_\ell \quad (\ell = 0, 1, \ldots)$
if $[x_0, \ldots, x_{\ell-1}]$ is a feasible solution
then {
  $P \leftarrow$ profit$([x_0, \ldots, x_{\ell-1}])$
  if $P > OptP$
  then {
    $OptP \leftarrow P$
    $OptX \leftarrow [x_0, \ldots, x_{\ell-1}]$
  }
}
Compute $\mathcal{C}_\ell$
$B \leftarrow$ B$([x_0, \ldots, x_{\ell-1}])$
for each $x \in \mathcal{C}_\ell$
do {
  if $B \leq OptP$ **then return**
  $x_\ell \leftarrow x$
  BOUNDING$(\ell + 1)$
}

BB aplicado ao problema do Caixeiro Viajante

Relembrando o algoritmo de backtracking para CAIXEIRO visto anteriormente:

Algorithm 4.10: TSP1 (ℓ)

global C_ℓ $\quad (\ell = 0, 1, \ldots, n-1)$
if $\ell = n$
then $\begin{cases} C \leftarrow \text{cost}([x_0, \ldots, x_{n-1}]) \\ \textbf{if } C < OptC \\ \quad \textbf{then} \begin{cases} OptC \leftarrow C \\ OptX \leftarrow [x_0, \ldots, x_{n-1}] \end{cases} \end{cases}$
if $\ell = 0$
then $C_\ell \leftarrow \{0\}$
else $\begin{cases} \textbf{if } \ell = 1 \\ \quad \textbf{then } C_\ell \leftarrow \{1, \ldots, n-1\} \\ \quad \textbf{else } C_\ell \leftarrow C_{\ell-1} \setminus \{x_{\ell-1}\} \end{cases}$
for each $x \in C_\ell$
do $\begin{cases} x_\ell \leftarrow x \\ \text{TSP1}(\ell + 1) \end{cases}$

Como CAIXEIRO é um problema de minimização, dada sol. parcial viável X, temos que

$B(X)$ é agora um **limitante inferior**, i.e., uma subestimativa para $P(X)$

BB aplicado ao problema do Caixeiro Viajante

Seja $x \in V(G)$ e $S \subseteq V(G)$, com $S \neq \emptyset$. Defina:

$$b(x, S) = \min\{w(xy) \mid y \in S \text{ e } y \neq x\}.$$

Teorema: Seja $X^* = [x_0, ..., x_{n-1}]$ um circuito hamiltoniado de custo mínimo em G. Seja $cost(X^*)$ o valor da função objetivo da solução X^*. Suponha que X^* que pode ser obtido da solução viável parcial $X = [x_0, ..., x_{l-1}]$, $l \leq n-1$, e seja $C_l = V \setminus \{x_0, ..., x_{l-1}\}$. Então

$$cost(X^*) \geq \sum_{i=0}^{l-2} w(x_i x_{i+1}) + b(x_{l-1}, C_l) + \sum_{y \in S} b(y, C_l \cup \{x_0\}).$$

A partir de $X = [x_0, ..., x_{l-1}]$, defina $B(X)$:

- Se $l < n-1$, então
  $B(X) = \sum_{i=0}^{l-2} w(x_i x_{i+1}) + b(x_{l-1}, C_l) + \sum_{y \in C_l} b(y, C_l \cup \{x_0\})$.
- Se $l = n-1$, então
  $B(X) = \sum_{i=0}^{l-2} w(x_i x_{i+1}) + w(x_{n-1} x_0)$

BB aplicado ao problema do Caixeiro Viajante

Algorithm 4.13: TSP2 (ℓ)

external B()
global C_ℓ $(\ell = 0, 1, \ldots, n-1)$
if $\ell = n$
then $\begin{cases} C \leftarrow \text{cost}([x_0, \ldots, x_{n=1}]) \\ \textbf{if } C < OptC \\ \quad \textbf{then} \begin{cases} OptC \leftarrow C \\ OptX \leftarrow [x_0, \ldots, x_{n-1}] \end{cases} \end{cases}$
if $\ell = 0$ **then** $C_\ell \leftarrow \{0\}$
else if $\ell = 1$ **then** $C_\ell \leftarrow \{1, \ldots, n-1\}$
else $C_\ell \leftarrow C_{\ell-1} \setminus \{x_{\ell-1}\}$
$B \leftarrow \text{B}([x_0, \ldots, x_{\ell-1}])$
for each $x \in C_\ell$
do $\begin{cases} \textbf{if } B \geq OptC \\ \quad \textbf{then return} \\ x_\ell \leftarrow x \\ \text{TSP2}(\ell + 1) \end{cases}$

BB aplicado ao problema do Caixeiro Viajante

Melhora Importante: Levando em consideração a ordenação das escolhas C_l:

Algorithm 4.23: TSP3 (ℓ)

external Sort(), cost()
global C_ℓ $(\ell = 0, 1, \ldots, n-1)$
if $\ell = n$
  then { $C \leftarrow \text{cost}([x_0, \ldots, x_{n-1}])$
    if $C < OptC$
      then { $OptC \leftarrow C$
        $OptX \leftarrow [x_0, \ldots, x_{n-1}]$
if $\ell = 0$ **then** $C_\ell \leftarrow \{0\}$
  else if $\ell = 1$ **then** $C_\ell \leftarrow \{1, \ldots, n-1\}$
  else $C_\ell \leftarrow C_{\ell-1} \setminus \{x_{\ell-1}\}$
$count \leftarrow 0$
for each $x \in C_\ell$
  do { $x_\ell \leftarrow x$
    $nextchoice[count] \leftarrow x$
    $nextbound[count] \leftarrow \text{B}([x_0, \ldots, x_{\ell-1}, x])$
    $count \leftarrow count + 1$
Sort *nextchoice* and *nextbound*
  so that *nextbound* is in increasing order
for $i \leftarrow 0$ **to** $count - 1$
  do { **if** $nextbound[i] \geq OptP$
    then return
    $x_\ell \leftarrow nextchoice[i]$
    TSP3($\ell + 1$)

BB com ordenação – Algoritmo geral

Melhora Importante: Levando em consideração a ordenação das escolhas C_l:

Algorithm 4.22: BRANCHANDBOUND (ℓ)

external B(), profit()
global C_ℓ $(\ell = 0, 1, \ldots)$
if $[x_0, \ldots, x_{\ell-1}]$ is a solution
then
- $P \leftarrow \text{profit}([x_0, \ldots, x_{\ell-1}])$
- **if** $P > OptP$
  - **then**
    - $OptP \leftarrow P$
    - $OptX \leftarrow [x_0, \ldots, x_{\ell-1}]$

Compute C_ℓ
$count \leftarrow 0$
for each $x \in C_\ell$
do
- $x_\ell \leftarrow x$
- $nextchoice[count] \leftarrow x$
- $nextbound[count] \leftarrow \text{B}([x_0, \ldots, x_{\ell-1}, x])$
- $count \leftarrow count + 1$

Sort *nextchoice* and *nextbound*
so that *nextbound* is in decreasing order
for $i \leftarrow 0$ **to** $count - 1$
do
- **if** $nextbound[i] \leq OptP$
  - **then return**
- $x_\ell \leftarrow nextchoice[i]$
- BRANCHANDBOUND($\ell + 1$)

Obs: No livro, embora seja incomum, apenas esta última versão da estratégia é chamada de Branch-and-Bound Para nós (e na literatura em geral), para se caracterizar Branch-and-Bound, o algoritmo não precisa disto.