CI1238 - Otimização

Aula 09 - Backtracking

Professor Murilo V. G. da Silva

Departamento de Informática

Universidade Federal do Paraná

21/03/2023

Backtracking

- Método para recursivamente explorar espaço de soluções
- Mais para frente iremos refinar o método para podemos "podarmos" ramos árvore de recursão que certamente não tem a solução procurada

Vamos ilustrar com um exemplo: Resolvendo o Problema da Mochila.

Mochila: solução recursiva

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 (capacidade da mochila).

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

Algorithm 4.1: KNAPSACK1 (ℓ)

```
global X, OptP, OptX
if ℓ = n
  then { if Σ_{i=0}^{n-1} w_i x_i ≤ M
           then { CurP ← Σ_{i=0}^{n-1} p_i x_i
                  if CurP > OptP
                    then { OptP ← CurP
                           OptX ← [x_0, ..., x_{n-1}] }
  else { x_ℓ ← 1
         KNAPSACK1(ℓ + 1)
         x_ℓ ← 0
         KNAPSACK1(ℓ + 1) }
```

Chamada inicial: $l = 0$

Mochila: solução recursiva

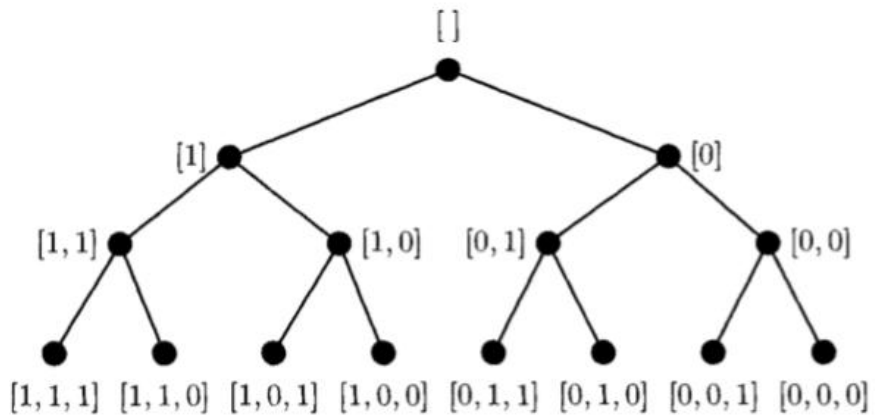
Algorithm 4.1: KNAPSACK1 (ℓ)

global X, $OptP$, $OptX$

if $\ell = n$

then
- **if** $\sum_{i=0}^{n-1} w_i x_i \leq M$
  - **then**
    - $CurP \leftarrow \sum_{i=0}^{n-1} p_i x_i$
    - **if** $CurP > OptP$
      - **then**
        - $OptP \leftarrow CurP$
        - $OptX \leftarrow [x_0, \ldots, x_{n-1}]$

else
- $x_\ell \leftarrow 1$
- KNAPSACK1($\ell + 1$)
- $x_\ell \leftarrow 0$
- KNAPSACK1($\ell + 1$)



Árvore de espaço de estados para $n = 3$

Algoritmo Genérico de Backtracking

Espaço de soluções como toda k-tupla $[x_0, x_1, ..., x_k]$, $k \leq n$. Aqui temos $x_i \in P_i$

- Seja $X^* = [x_0, x_1, ..., x_k]$ uma solução viável para um problema de otimização.
- Solução (viável) parcial: uma lista $[x_0, x_1, ..., x_l]$, $l \leq k$

  No livro às vezes a notação também é $[x_0, x_1, ...]$
- Note: a partir de $[x_0, ..., x_{l-1}]$ e das restrições do problema em particular:
  - x_l pode assumir apenas um conjunto restrito de valores $C_l \subseteq P_l$

    O conjunto C_l é o conjunto de escolhas para x_l
  - A computação C_l é chamado de poda da árvore

    Se $y \in P_l \setminus C_l$, nós da árvore **NÃO** tem $[x_0, ..., x_{l-1}, y]$ como sol. parcial

Algorithm 4.2: BACKTRACK (ℓ)

global $X, C_\ell \quad (\ell = 0, 1, \ldots)$
comment: $X = [x_0, x_1, \ldots]$
if $[x_0, x_1, \ldots, x_{\ell-1}]$ is a feasible solution
  then process it
Compute C_ℓ
for each $x \in C_\ell$
  do $\begin{cases} x_\ell \leftarrow x \\ \text{BACKTRACK}(\ell + 1) \end{cases}$

No algoritmo acima, "processa" depende do problema específico.

Backtracking para Mochila

Algorithm 4.3: KNAPSACK2 $(\ell, CurW)$

global $X, OptX, OptP, C_\ell \quad (\ell = 0, 1, \ldots)$

if $\ell = n$

then $\begin{cases} \textbf{if } \sum_{i=0}^{n-1} p_i x_i > OptP \\ \quad \textbf{then} \begin{cases} OptP \leftarrow \sum_{i=0}^{n-1} p_i x_i \\ OptX \leftarrow [x_0, \ldots, x_{n-1}] \end{cases} \end{cases}$

if $\ell = n$

then $C_\ell \leftarrow \emptyset$

else $\begin{cases} \textbf{if } CurW + w_\ell \leq M \\ \quad \textbf{then } C_\ell \leftarrow \{1, 0\} \\ \quad \textbf{else } C_\ell \leftarrow \{0\} \end{cases}$

for each $x \in C_\ell$

do $\begin{cases} x_\ell \leftarrow x \\ \text{KNAPSACK2}(\ell + 1, CurW + w_\ell x_\ell) \end{cases}$

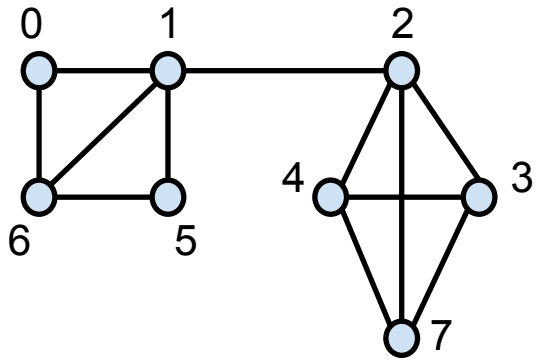
A primeira chamada do algoritmo é feita com $l = CurW = 0$.

Backtracking para Problema das cliques maximais

Veremos agora uma algoritmo que toma um grafo de entrada e:

- Enumera todas as suas cliques
- Identifica entre estas quais cliques são maximais

O algoritmo pode também ser facilmente adaptado para identificar a clique máxima



Cliques do grafo do exemplo acima:

- Tamanho 0: $\{\}$ (livro considera tal caso)
- Tamanho 1: $\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}$
- Tamanho 2: $\{0,1\}, \{0,6\}, \{1,2\}, \{1,5\}, \{1,6\}, \{2,4\}, \{2,3\}, \{2,7\}, \{3,4\}, \{3,7\}, \{4,7\}, \{5,6\}$
- Tamanho 3: $\{0,1,6\}, \{1,5,6\}, \{2,3,4\}, \{2,3,7\}, \{2,4,7\}$ $\{3,4,7\}$
- Tamanho 4: $\{2,3,4,7\}$

Cliques maximais: $\{1,2\}$, $\{0,1,6\}, \{1,5,6\}$, $\{2,3,4,7\}$
Clique máxima: $\{2,3,4,7\}$

Backtracking para Problema das cliques maximais

Gerando todas as cliques do grafo usando *backtracking*:

- O que é uma solução parcial e como computar C_l aqui?
  - Solução parcial: $[x_0, ..., x_{l-1}]$ é solução parcial se $\{x_0, ..., x_{l-1}\}$ é clique
  - Computando C_l:
    - Seja $C_0 = V(G)$ i.e., No passo 0 todo vértice podem ser escolhidos
    - Seja $S_{l-1} = \{x_0, ..., x_{l-1}\}$ (vértices da clique solução parcial)
    - $C_l = \{v \in V(G) \setminus S_{l-1} \mid \{v, x\} \in E(G)$, para cada $x \in S_{l-1}\}$.

Backtracking para Problema das cliques maximais

Do slide anterior:

- $C_l = \{v \in V(G) \setminus S_{l-1} \mid \{v, x\} \in E(G), \text{ para cada } x \in S_{l-1}\}$

Note que podemos reescrever C_l da seguinte maneira:

$$C_l = \{v \in C_{l-1} \setminus \{x_{l-1}\} \mid \{v, x_{l-1}\} \in E(G)\}$$

- Escolha do l-ésimo vértice: temos um escolha a menos que na escolha anterior.
- Mais precisamente, no vértice escolhido anteriormente não está mais disponível.

Problema com esta abordagem: Cada clique de tamanho k será gerada $k!$ vezes

Ideia: Criamos uma ordenação (arbitrária) dos vértices do grafo:

- Lista com vértices do grafo: $[v_0, v_1, ..., v_n]$ tal que $v_1 < v_2 < \cdots < v_n$

Reescrevendo (novamente!) as escolhas:

$$C_l = \{v \in C_{l-1} \mid \{v, x_{l-1}\} \in E(G) \text{ e } v > x_{l-1}\}$$

Note:

- Podemos fazer $\{v \in C_{l-1}$, pois a condição $\{v, x_{l-1}\} \in E(G)$ garante que o vértice x_{l-1} não está disponível (não existem laços!)
- A condição $v > x_{l-1}$ garante que cada clique apareça um vez (vértices são inseridos na clique na ordem)

Backtracking para Problema das cliques maximais

Do slide anterior, as escolhas para o l-ésimo vértice da clique é:

$$C_l = \{v \in C_{l-1} \mid \{v, x_{l-1}\} \in E(G) \text{ e } v > x_{l-1}\}$$

Notação:

- $A_v = \{u \in V(G) \mid \{u, v\} \in E(G)\}$ (notação no livro para vizinhança de v)
- $B_v = \{u \in V(G) \mid u > v\}$ (vértices com índices maiores)
- Obs: A_v e B_v podem ser computados em pré-processamento

Reescrevendo C_l pela última vez(!):

$$C_l = A_{x_{l-1}} \cap B_{x_{l-1}} \cap C_{l-1}$$

Seja $X = [x_0, ..., x_{l-1}]$ uma clique de G.

- Úlitma notação auxiliar: $N_0 = V(G)$; $N_l = N_{l-1} \cap A_{x_{l-1}}$
- Note: X é clique maximal $\Leftrightarrow N_l = \emptyset$.

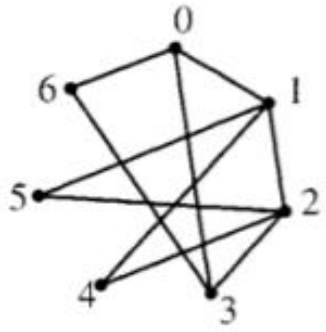
Backtracking para Problema de Listar Cliques

Algorithm 4.4: ALLCLIQUES (ℓ)

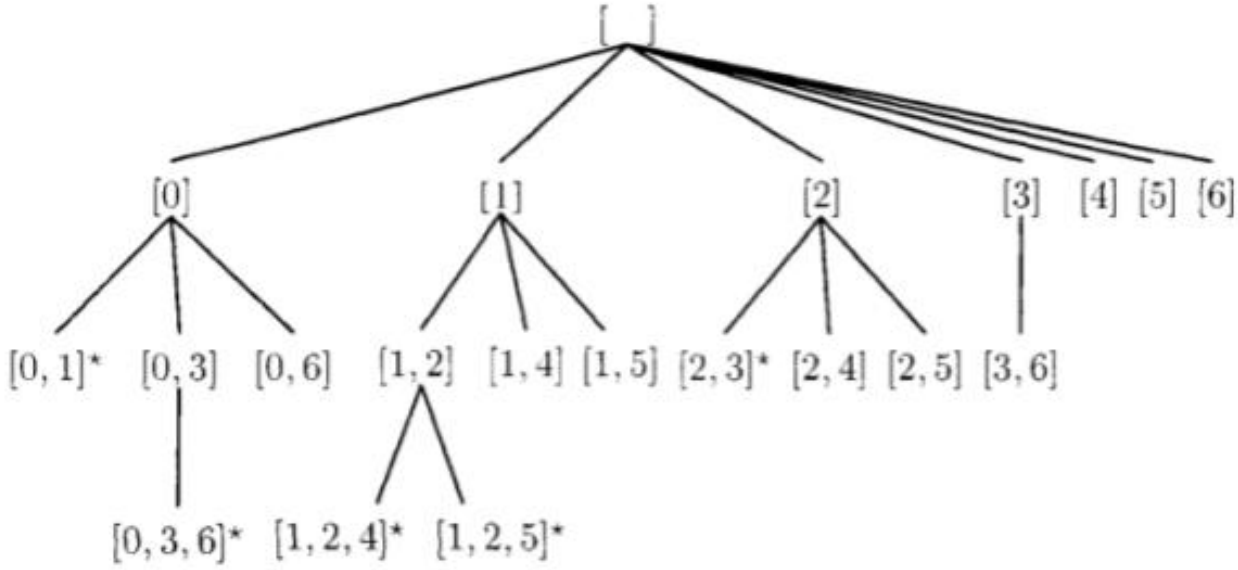
global $X, \mathcal{C}_\ell \quad (\ell = 0, 1, \ldots, n-1)$
comment: every X generated by the algorithm is a clique
if $\ell = 0$
  then output $([\,])$
  else output $([x_0, \ldots, x_{\ell-1}])$
if $\ell = 0$
  then $N_\ell \leftarrow \mathcal{V}$
  else $N_\ell \leftarrow A_{x_{\ell-1}} \cap N_{\ell-1}$
if $N_\ell = \emptyset$
  then $\{x_0, \ldots, x_{\ell-1}\}$ is a maximal clique
if $\ell = 0$
  then $\mathcal{C}_\ell \leftarrow \mathcal{V}$
  else $\mathcal{C}_\ell \leftarrow A_{x_{\ell-1}} \cap B_{x_{\ell-1}} \cap \mathcal{C}_{\ell-1}$
for each $x \in \mathcal{C}_\ell$
  do $\begin{cases} x_\ell \leftarrow x \\ \text{ALLCLIQUES}(\ell + 1) \end{cases}$

A primeira chamada do algoritmo é feita com $l = 0$.

Backtracking para Problema de Listar Cliques



v	A_v	B_v
0	1, 3, 6	1, 2, 3, 4, 5, 6
1	0, 2, 4, 5	2, 3, 4, 5, 6
2	1, 3, 4, 5	3, 4, 5, 6
3	0, 2, 6	4, 5, 6
4	1, 2	5, 6
5	1, 2	6
6	0, 3	



Limitantes - Branch-and-Bound

Até agora: Podamos da árvore os ramos com **soluções inviáveis**.

- Que outros ramos podemos podar?
  - Ramos que garantidamente não tem solução ótima.
- Especificamente: Dada melhor solução encontrada até o momento.
  - A estratégia é descartar ramos que garantidamente não tem (nem mesmo) solução melhor do que a melhor já encontrada.

Formalizando:

- Dado problema de maximização com função objetivo $p()$ e solução ótima OPT.
- Seja $X = [x_0, ..., x_{l-1}]$ uma solução viável **parcial**
  - $P(X) = \max\{p(X') \mid X' \text{ é solução viável descendente na árvore de busca}\}$

    i.e. $P(X) = \max\{p(X') \mid X' = [x'_0, ..., x'_{l-1}, ..., x'_k]$ é solução viável e $x_i = x'_i$ para $i = 0, ..., l-1\}$
  - Note: Se $X = [\,]$, então $P(X) = OPT$
- Em geral, computar $P()$ é custoso
- Ideia: Encontrar função $B()$ tal que:
  - Para toda solução parcial viável, $P(X) \leq B(X)$. i.e., um limitante superior
  - Note: Caso $B(X) \leq OPT$, então $P(X) \leq B(X) \leq OPT$

Limitantes - Branch-and-Bound

Algorithm 4.7: BOUNDING (ℓ)

external $\mathsf{P}(), \mathsf{B}()$

global $X, OptP, OptX, \mathcal{C}_\ell \quad (\ell = 0, 1, \ldots)$

if $[x_0, \ldots, x_{\ell-1}]$ is a feasible solution

then
- $P \leftarrow \mathsf{profit}([x_0, \ldots, x_{\ell-1}])$
- **if** $P > OptP$
  - **then**
    - $OptP \leftarrow P$
    - $OptX \leftarrow [x_0, \ldots, x_{\ell-1}]$

Compute $\mathcal{C}_\ell$

$B \leftarrow \mathsf{B}([x_0, \ldots, x_{\ell-1}])$

for each $x \in \mathcal{C}_\ell$

do
- **if** $B \leq OptP$ **then return**
- $x_\ell \leftarrow x$
- BOUNDING$(\ell + 1)$

Mochila Fracionário

Queremos aplicar *BB* no problema MOCHILA, mas antes considere o problema:

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 $\sum_{i=0}^{n-1} x_i w_i \leq C$ e $\sum_{i=0}^{n-1} x_i v_i$ seja máximo.

Algoritmo Guloso:

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 $[x_0, ..., x_{l-1}]$, defina:

$$\begin{aligned} \mathrm{B}(X) &= \sum_{i=0}^{\ell-1} p_i x_i + \textsc{RKnap}(p_\ell, \ldots, p_n, w_\ell, \ldots, w_n, M - \sum_{i=0}^{\ell-1} w_i x_i) \\ &= \sum_{i=0}^{\ell-1} p_i x_i + \textsc{RKnap}(p_\ell, \ldots, p_n, w_\ell, \ldots, w_n, M - CurW) \end{aligned}$$

Portanto, $B(X)$ é a soma

- (1) dos valores obtidos pelos itens $0, ..., l-1$
- (2) mais o valor para itens restantes na *versão fracionária do problema* para o espaço que sobrou na mochila, i.e., $M - CurW$.
- Note:
  - Se fizermos x_i em (2) ser 0 ou 1, temos $P(X) = B(X)$
  - Com $x_i \in \mathbb{R}$ a estimativa é mais frouxa, mas computável eficientemente.

BB aplicado ao problema da Mochila

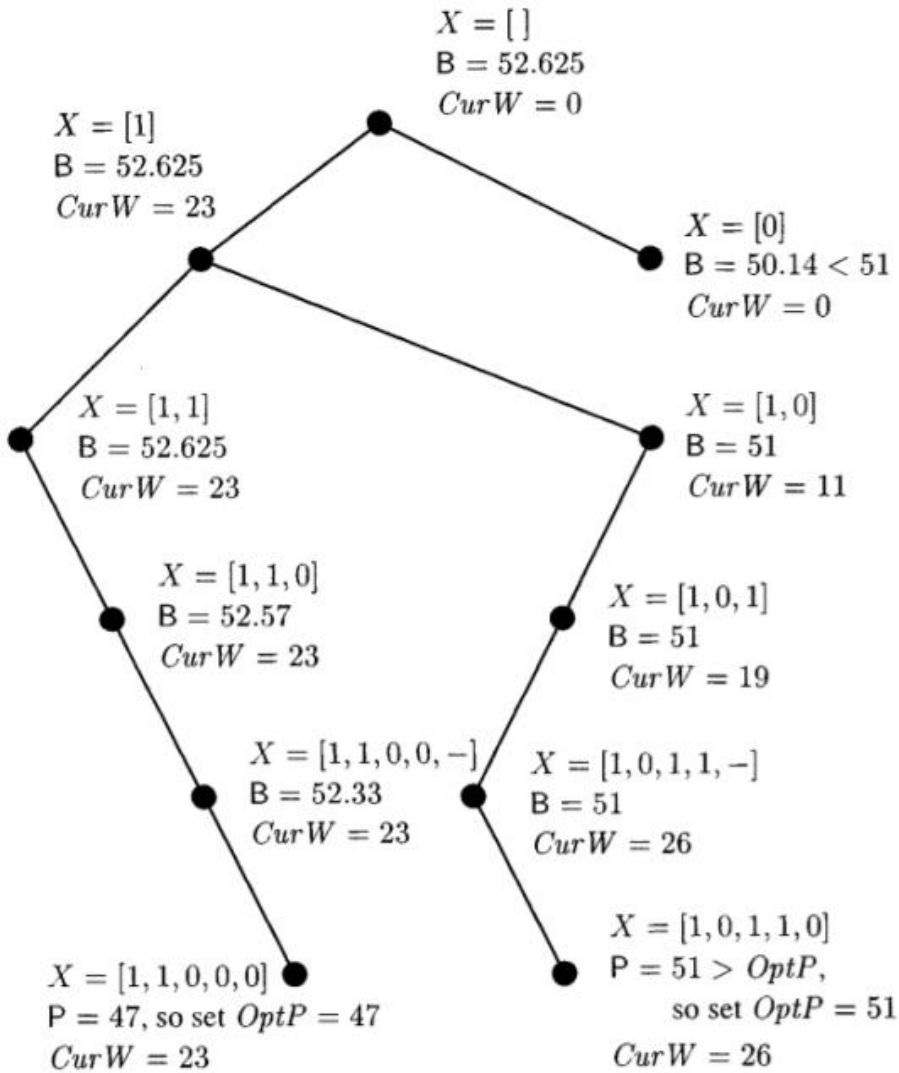
Para o algoritmo abaixo, assumimos que os itens estão ordenados tal que $\frac{p_i}{w_i} \leq \frac{p_{i+1}}{w_{i+1}}$

Obs: Com isso, podemos remover a ordenação em RKNAP, se quisermos

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$(\ell + 1, CurW + w_\ell x_\ell)$

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

BB aplicado ao problema do Caixeiro Viajante

Considere um grafo ponderado com $V(G) = \{0, 1, ..., N-1\}$.

- Queremos encontrar um caminho hamiltoniado de custo mínimo em G

Como aqui temos um problema de minimização, dada solução parcial viável X

$B(X)$ é agora um limitante inferior!

Vamos obter um Algoritmo de Branch-and-Bound para o problema

- Mas primeiramente, considere a solução mais simples usando *backtracking*:

BB aplicado ao problema do Caixeiro Viajante

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 $\begin{cases} x_\ell \leftarrow x \\ \text{TSP1}(\ell + 1) \end{cases}$

BB aplicado ao problema do Caixeiro Viajante

Seja $x \in V(G)$ e $W \subseteq V(G)$, com $W \neq \emptyset$. Seja $cost(\{x, y\})$ a função para os pesos das arestas de G. Defina:

$$b(x, W) = \min\{cost(x, y)\ y \in W \text{ e } y \neq x\}.$$

Teorema: Seja $X^* = [x_0, ..., x_{N-1}]$ um circuito hamiltoniado de peso mínimo em G que pode ser obtido da solução viável parcial $X = [x_0, ..., x_{l-1}]$, $l \leq N - 1$. Então

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

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

- Se $l \leq N - 1$, então $B(X) = \sum_{i=0}^{l-2} cost(x_i, x_{i+1}) + b(x_{l-1}, \{x_l, ..., x_{N-1}\}) + \sum_{y \in \{x_l, ... x_{N-1}\}=0} b(y, \{x_l, ..., x_{N-1}, x_0\})$.
- Se $l = N - 1$, então $B(X) = \sum_{i=0}^{l-2} cost(x_i, x_{i+1}) + cost(x_{n-1}, x_0)$

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:

Obs: No livro, embora seja incomum, apenas esta última versão da estratégia é chamada de Branch-and-Bound

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($\ell + 1$)

BB aplicado ao problema do Caixeiro Viajante

TSP usando ordenação das escolhas:

Algorithm 4.23: TSP3 (ℓ)

external Sort(), cost()
global $C_\ell \quad (\ell = 0, 1, \ldots, n-1)$
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 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 $\begin{cases} x_\ell \leftarrow x \\ nextchoice[count] \leftarrow x \\ nextbound[count] \leftarrow \mathsf{B}([x_0, \ldots, x_{\ell-1}, x]) \\ count \leftarrow count + 1 \end{cases}$
Sort *nextchoice* and *nextbound*
  so that *nextbound* is in increasing order
for $i \leftarrow 0$ **to** $count - 1$
  do $\begin{cases} \textbf{if } nextbound[i] \geq OptP \\ \quad \textbf{then return} \\ x_\ell \leftarrow nextchoice[i] \\ \text{TSP3}(\ell + 1) \end{cases}$