# Algoritmos Combinatórios: Backtracking com "Bounding"

Lucia Moura
lucia@site.uottawa.ca

UFSC, Fevereiro, 2010

# Backtracking com "bounding"

Quando aplicamos backtracking para um problema de **otimização**, usamos **bounding** para podar a árvore de espaço de estados.
Consiremos um problema de **maximização**.
Seja $\text{lucro}(X) =$ o lucro de uma solução viável $X$.
Para um solução parcial $X = [x_0, x_1, \ldots, x_{l-1}]$, defina

$$P(X) = \max \quad \{ \quad \text{lucro}(X') : X' = [x_0, x_1, \ldots, x_{l-1}, x'_l, \ldots, x'_{n-1}] \text{ e' uma solucao viavel } \}.$$

Uma **função de bounding** $B$ é uma função de valores reais definida nos vértices da árvore de epaço de estadps, tais que para toda solução viável $X$, $B(X) \geq P(X)$.
$B(X)$ é um limite superior (upper bound) do lucro de qualquer solução que seja descendente de $X$ na árvore de espaço de estados.
**Se a melhor solução encontrada até o momento tem valor $OptP$, então podemos podar vértices $X$ com $B(X) \leq OptP$.**

# Algoritmo Genal de Backtracking com "Bounding"

Algoritmo $\textsc{BacktrackBounding}(l)$
Global $X$, $OptP$, $OptX$, $\mathcal{C}_l$, $l = (0, 1, \ldots)$
if ($[x_0, x_1, \ldots, x_{l-1}]$ e' uma solution viavel) then
  $P \leftarrow \text{profit}([x_0, x_1, \ldots, x_{l-1}])$;
  if $(P > OptP)$ then
    $OptP \leftarrow P$;
    $OptX \leftarrow [x_0, x_1, \ldots, x_{l-1}]$;
Compute $\mathcal{C}_l$;
$B \leftarrow B([x_0, x_1, \ldots, x_{l-1}])$;
for each $(x \in \mathcal{C}_l)$ do
  if $B \leq OptP$ then return;
  $x_l \leftarrow x$;
  $\textsc{BacktrackBounding}(l + 1)$

# Problema da Mochila Racional

Usaremos uma solução do Problema da Mochila Racional como função de bounding para o Problema da Mochila (inteiro/binário).

Problema da Mochila Racional:

Instância: Lucros $p_0, p_1, \ldots, p_{n-1}$
Pesos $w_0, w_1, \ldots, w_{n-1}$
Capacidade da mochila $M$

Encontre: uma $n$-tupla $[x_0, x_1, \ldots, x_{n-1}]$
tal que o lucro total $P = \sum_{i=0}^{n-1} p_i x_i$ seja maximizado,
sujeito a restrição de capacidade: $\sum_{i=0}^{n-1} w_i x_i \leq M$.
onde $x_i$ e' racional e $0 \leq x_i \leq 1$, pra todo $i = 0, \ldots, n-1$.

# Solução da Mochila Racional

Algoritmo MochilaRacional $((p_0, \ldots, w_{n-1}), (w_0, \ldots, w_{n-1}), M)$

Permute os indices de modo que $p_0/w_0 \geq p_1/w_1 \geq \cdots \geq p_{n-1}/w_{n-1}$

$i \leftarrow 0$; $P \leftarrow 0$; $W \leftarrow 0$

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 $P$

**Teorema**

*O Algoritmo* MochilaRacional *retorna o valor objectivo de uma solução ótima do problema da mochila racional.*

# Função de Bounding para o problem da Mochila

Dada uma solução parcial (viável) $X = [x_0, x_1, \ldots, x_{l-1}]$, defina:

$$B(X) = \sum_{i=0}^{l-1} p_i x_i + \\ + \textsc{MochilaRacional}((p_l, \ldots, p_{n-1}), (w_l, \ldots, w_{n-1}), M - \sum_{i=0}^{l-1} w_i x_i)$$

$B(X)$ é a soma de:

1. o lucro obtido com objectos $0, 1, \ldots, l-1$, mais
2. o lucro obtido dos objetos restantes, usando a capacidade restante $M - \sum_{i=0}^{l-1} w_i x_i$, mas permitindo valores de $x_i$ racionais (para $l \leq i \leq n-1$) com $0 \leq x_i \leq 1$.

Se na parte 2. restringíssemos $x_i$ a valores binários, obteriamos $P(X)$, mas permitindo valores racionais para $x_l, \ldots, x_{n-1}$ pode resultar num lucro maior; ou seja, $B(X) \geq P(X)$ e $B$ é um função de bounding.

# Backtracking com Bounding para o Problema da Mochila

Variaveis Globais $X, OptP, OptX$.

Algoritmo MOCHILA3 $(l, CurW)$

if $(l = n)$ then if $(\sum_{i=0}^{n-1} p_i x_i > OptP)$ then
$OptP \leftarrow \sum_{i=0}^{n-1} p_i x_i$;
$OptX \leftarrow [x_0, x_1, \ldots, x_{n-1}]$;

if $(l = n)$ then $\mathcal{C}_l \leftarrow \emptyset$

else if $(CurW + w_l \leq M)$ then $\mathcal{C}_l \leftarrow \{0, 1\}$;
else $\mathcal{C}_l \leftarrow \{0\}$;

$B \leftarrow \sum_{i=0}^{l-1} p_i x_i +$
MOCHILARACIONAL$((p_l, \ldots, p_{n-1}), (w_l, \ldots, w_{n-1}), M - CurW)$

for each $x \in \mathcal{C}_l$ do
if $B \leq OptP$ then return;
$x_l \leftarrow x$;
MOCHILA3 $(l + 1, CurW + w_l x_l)$;

# Tamanho da árvore de espaço de estados p/ Mochila

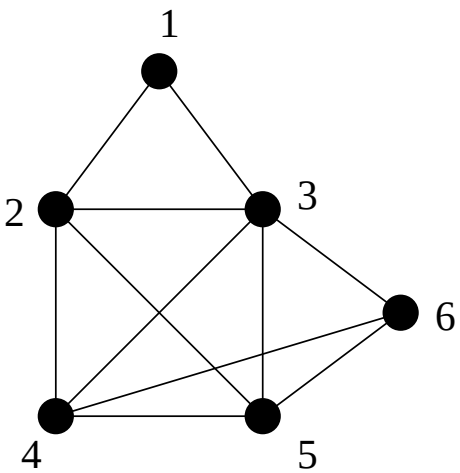
| n | Alg. MOCHILA1 | Alg. MOCHILA2 | Alg. MOCHILA3 |
|---|---|---|---|
| 8 | 511 | 322 | 67 |
| 12 | 8191 | 4804 | 130 |
| 16 | 131071 | 75936 | 282 |
| 20 | 2097151 | 1182784 | 283 |
| 24 | 33554431 | 18963596 | 493 |

Valor médio obtido em 5 problemas de mochilas aleatórias de tamanho $n$.

# Problema do Clique Máximo

PROBLEMA: Clique Máximo
INSTÂNCIA: grafo $G = (V, E)$.
ENCONTRE: um clique de cardinalidade máxima em $G$.

Este problema é NP-difícil.




Maximum cliques:

{2,3,4,5}, {3,4,5,6}

Modificação de TodosCliques para clique máximo (sem "bounding").
Linhas em azul adicionam **"bounding"** a este algoritmo.

Algoritmo $\text{MaxClique}(l)$
Global: $X$, $\mathcal{C}_l(l = 0, \ldots, n-1)$, $A_l$, $B_l$ pre-computado.

```
        if (l > OptSize) then
           OptSize ← l;
           OptClique ← [x_0, x_1, ..., x_{l-1}];
        if (l = 0) then C_l ← V;
                   else C_l ← A_{x_{l-1}} ∩ B_{x_{l-1}} ∩ C_{l-1};
        M ← B([x_0, x_1, ..., x_{l-1}]);
        for each (x ∈ C_l) do
            if (M ≤ OptSize) then return;
            x_l ← x; MaxClique(l + 1);
```

Main

```
    OptSize ← 0; MaxClique(0);
    output OptClique;
```

# Funções de "Bounding" para MaxClique

## Definição (Subgrafo induzido)

*Seja $G = (V, E)$ e $W \subseteq V$. O* subgrafo induzido *por $W$, $G[W]$, tem conjunto de vértices $W$ e conjunto de arestas: $\{\{u, v\} \in E : u, v \in W\}$.*

solução parcial: $X = [x_0, x_1, \ldots, x_{l-1}]$ com conjunto de escolha $\mathcal{C}_l$,
solução de extensão $X = [x_0, x_1, \ldots, x_{l-1}, x_l, \ldots, x_j]$,

Então $\{x_l, \ldots, x_j\}$ é um clique de $G[\mathcal{C}_l]$.
Seja $mc(l)$ denote o tamanho do clique máximo em $G[\mathcal{C}_l]$, e seja $ub(l)$ um limite superior de $mc(l)$.
Então, uma função de "bounding" é $B(X) = l + ub(l)$.

# Bound baseado no tamanho do subgrafo

Função geral de bounding:

$$B(X) = l + ub(l)$$

Como $mc(l) \leq |\mathcal{C}_l|$, temos a seguinte função de bounding:

$$B_1(X) = l + |\mathcal{C}_l|.$$

# Bound baseado em coloração do grafo

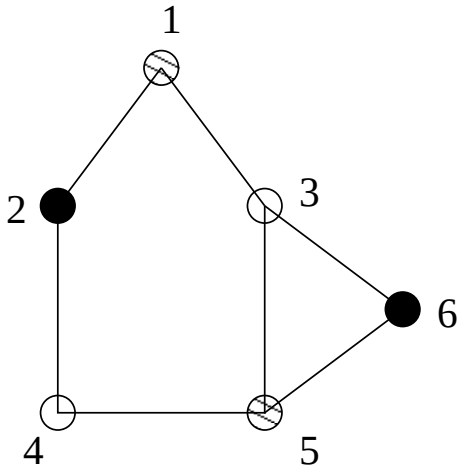
## Definição (Coloração de vértices)

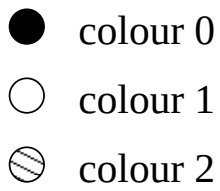
*Sejam um grafo $G = (V, E)$ e um inteiro positivo $k$. Uma $k$-coloração (de vértices) de $G$ é uma função*

$$\text{COR}: V \to \{0, 1, \ldots, k-1\}$$

*tal que, para toda aresta $\{x, y\} \in E$, $\text{COR}(x) \neq \text{COR}(y)$.*

Exemplo: uma 3-coloração de um grafo:



**Lema**

*Se $G$ tem uma $k$-coloração, então um clique máximo de $G$ tem cardinalidade $\leq k$.*

**Prova.** Seja um clique $C$. Cada $x \in C$ tem que ter uma cor distinta. Então, $|C| \leq k$. Isto vale para qualquer clique, em particular para um clique máximo. $\square$

Encontrar uma coloração de cardinalidade mínima daria o melhor limite superior, mas é um problema difícil. Usamos um **algoritmo guloso** pra encontrar uma coloração pequena.
Defina $\textsc{ClasseDeCor}[h] = \{i \in V : \textsc{Cor}[i] = h\}$.

$\textsc{ColorirGuloso}(G = (V, E))$
Global $\textsc{Cor}$
$k \leftarrow 0$; // cores usadas ate agora
for $i \leftarrow 0$ to $n - 1$ do
$h \leftarrow 0$;
while $(h < k)$ and $(A_i \cap \textsc{ClasseDeCor}[h] \neq \emptyset)$ do
$h \leftarrow h + 1$;
if $(h = k)$ then $k \leftarrow k + 1$;
$\textsc{ClasseDeCor}[h] \leftarrow \emptyset$;
$\textsc{ClasseDeCor}[h] \leftarrow \textsc{ClasseDeCor}[h] \cup \{i\}$;
$\textsc{Cor}[i] = h$;
return $k$;

**Bound de Amostragem:**
Estaticamente, a priori, rode $\textsc{ColorirGuloso}(G)$, determinando $k$ e $\textsc{Cor}[x]$ para todo $x \in V$.

$\textsc{BoundDeAmostragem}(X = [x_0, x_1, \ldots, x_{l-1}])$
    Global $\mathcal{C}_l$, $\textsc{Cor}$
    return $l + |\{\textsc{Cor}[x] : x \in \mathcal{C}_l\}|$;

**Bound Guloso:**
Rode $\textsc{ColorirGuloso}$ dinamicamente.

$\textsc{BoundGuloso}(X = [x_0, x_1, \ldots, x_{l-1}])$
    Global $\mathcal{C}_l$
    $k \leftarrow \textsc{ColorirGuloso}(G[\mathcal{C}_l])$;
    return $l + k$;

Tamanho da árvore de busca; grafos aleatórios, densidade de arestas 0.5

| # vértices | 50 | 100 | 150 | 200 | 250 |
|---|---|---|---|---|---|
| # arestas | 607 | 2535 | 5602 | 9925 | 15566 |
| tamanho max clique | 7 | 9 | 10 | 11 | 11 |
| função de bounding: | | | | | |
| nenhuma | 8687 | 257145 | 1659016 | 7588328 | 26182672 |
| bound de tamanho | 3202 | 57225 | 350310 | 1434006 | 5008757 |
| bound de amostragem | 2268 | 44072 | 266246 | 1182514 | 4093535 |
| bound guloso | 430 | 5734 | 22599 | 91671 | 290788 |

# Branch-and-bound

Branch-and-Bound é uma generalização de backtracking com "bounding" em que a árvore de espaço de estados pode ser explorada de maneiras diferentes, tais como:

1. escolher uma ordem de exploração dos filhos de um vértice, dando prioridade a explorar filhos com bound maior (mais promissores em termos de função objetivo);
2. explorar os vértices não necessariamente de forma "depth-first" (pre-ordem) como em backtracking; em vez disso, manter uma fila de prioridades com vértices ativos (prioridade dada a vértices de maior bound), processando os vértices mais prioritários primeiro, e inserindo os seus filhos na fila de prioridades.

O algoritmo seguinte segue a idéia do item 1.

Algoritmo BranchAndBound($l$)

externo $B()$, LUCRO(); global $\mathcal{C}_l$ $(l = 0, 1, \ldots)$

if ($[x_0, x_1, \ldots, x_{l-1}]$ e' uma solucao viavel) then

$P \leftarrow$Lucro($[x_0, x_1, \ldots, x_{l-1}]$)

if $(P > OptP)$ then $OptP \leftarrow P$;

$OptX \leftarrow [x_0, x_1, \ldots, x_{l-1}]$;

Compute $\mathcal{C}_l$; $count \leftarrow 0$;

for each $(x \in \mathcal{C}_l)$ do

$nextchoice[count] \leftarrow x$;

$nextbound[count] \leftarrow B([x_0, x_1, \ldots, x_{l-1}, x])$;

$count \leftarrow count + 1$;

Ordene $nextchoice$ e $nextbound$ em ordem decrescente de $nextbound$;

for $i \leftarrow 0$ to $count - 1$ do

if $(nextbound[i] \leq OptP)$ then return;

$x_l \leftarrow nextchoice[i]$;

BranchAndBound$(l + 1)$;