CIC111

Pedro Hokama

1 de Junho de 2020

Problema[1]

Uma agroindústria deseja produzir uma ração.

- Três ingredientes básicos: osso, soja e peixe.
- Cada ingrediente tem diferentes níveis de dois nutrientes básicos: proteína e cálcio.
- O nutricionista exige que 1kg de ração tenha uma quantidade mínima desses nutrientes.
- Cada ingrediente é adquirido a um certo custo (\$/kg).
- Queremos determinar em que quantidades os ingredientes devem ser misturados para minimizar os custos.

	Osso	Soja	Peixe	Ração
Proteína	0.2	0.5	0.4	0.3
Cálcio	0.6	0.4	0.4	0.5
Custos \$/kg	0.56	0.81	0.46	

[1] Arenales, Marcos Nereu, et al. "Pesquisa operacional."(2007).

	Osso	Soja	Peixe	Ração
Proteína	0.2	0.5	0.4	0.3
Cálcio	0.6	0.4	0.4	0.5
Custos \$/kg	0.56	0.81	0.46	

- Queremos descobrir quanto de cada ingrediente precisamos colocar em 1kg de ração. Vamos definir essas quantidades como:
- x_{osso} é a quantidade de osso que vamos colocar para 1kg de ração.
- x_{soja} é a quantidade de soja que vamos colocar para 1kg de ração.
- x_{peixe} é a quantidade de peixe que vamos colocar para 1kg de ração.

$$
\begin{aligned}
\text{minimizar} \quad & 0.56x_{osso} + 0.81x_{soja} + 0.46x_{peixe}, \\
\text{sujeito a} \quad & 0.2x_{osso} + 0.5x_{soja} + 0.4x_{peixe} \geq 0.3, \\
& 0.6x_{osso} + 0.4x_{soja} + 0.4x_{peixe} \geq 0.5, \\
& x_{osso} + x_{soja} + x_{peixe} = 1, \\
& x_{osso} \geq 0, \quad x_{soja} \geq 0, \quad x_{peixe} \geq 0.
\end{aligned}
$$

Programação Linear

$$
\begin{aligned}
\text{minimizar} \quad & 0.56x_{osso} + 0.81x_{soja} + 0.46x_{peixe}, \\
\text{sujeito a} \quad & 0.2x_{osso} + 0.5x_{soja} + 0.4x_{peixe} \geq 0.3, \\
& 0.6x_{osso} + 0.4x_{soja} + 0.4x_{peixe} \geq 0.5, \\
& x_{osso} + x_{soja} + x_{peixe} = 1, \\
& x_{osso} \geq 0, \quad x_{soja} \geq 0, \quad x_{peixe} \geq 0.
\end{aligned}
$$

Um problema com

- Uma função objetivo linear,
- Restrições lineares e
- Variáveis continuas,

é conhecido como um **programa linear**.

Programação Linear

$$
\begin{aligned}
\text{minimizar} \quad & 0.56x_{osso} + 0.81x_{soja} + 0.46x_{peixe}, \\
\text{sujeito a} \quad & 0.2x_{osso} + 0.5x_{soja} + 0.4x_{peixe} \geq 0.3, \\
& 0.6x_{osso} + 0.4x_{soja} + 0.4x_{peixe} \geq 0.5, \\
& x_{osso} + x_{soja} + x_{peixe} = 1, \\
& x_{osso} \geq 0, \quad x_{soja} \geq 0, \quad x_{peixe} \geq 0.
\end{aligned}
$$

- Muitos problemas podem ser modelados como um programa linear.
- Existem algoritmos eficientes para resolver esses problemas.
- A melhor solução possível, chamada **solução ótima** ,

Programação Linear

- Existem algoritmos polinomiais para resolver Programação Linear.
- Curiosamente o algoritmo mais utilizado é o Simplex, que é exponencial mas muito eficiente na prática.
- Esses algoritmos não estão no escopo dessa disciplina.
- Existem diversos resolvedores capazes de resolver um programa linear.

Programação Linear

$$
\begin{aligned}
\text{minimizar} \quad & 0.56x_{osso} + 0.81x_{soja} + 0.46x_{peixe}, \\
\text{sujeito a} \quad & 0.2x_{osso} + 0.5x_{soja} + 0.4x_{peixe} \geq 0.3, \\
& 0.6x_{osso} + 0.4x_{soja} + 0.4x_{peixe} \geq 0.5, \\
& x_{osso} + x_{soja} + x_{peixe} = 1, \\
& x_{osso} \geq 0, \quad x_{soja} \geq 0, \quad x_{peixe} \geq 0.
\end{aligned}
$$

- Uma solução ótima deste modelo é $x^*_{osso} = 0.5$, $x^*_{soja} = 0$ e $x^*_{peixe} = 0.5$, cujo custo é $0.56 * 0.5 + 0.81 * 0 + 0.46 * 0.5 = 0.51$.
- Você pode explorar e tentar encontrar uma solução ótima. Mas será que podemos provar que essa solução é ótima?

$$
\begin{aligned}
\text{minimizar} \quad & 0.56x_{osso} + 0.81x_{soja} + 0.46x_{peixe}, \\
\text{sujeito a} \quad & 0.2x_{osso} + 0.5x_{soja} + 0.4x_{peixe} \geq 0.3, \\
& 0.6x_{osso} + 0.4x_{soja} + 0.4x_{peixe} \geq 0.5, \\
& x_{osso} + x_{soja} + x_{peixe} = 1, \\
& x_{osso} \geq 0, \quad x_{soja} \geq 0, \quad x_{peixe} \geq 0.
\end{aligned}
$$

- Em uma solução viável para o problema:

$$
\begin{aligned}
0.2x_{osso} + 0.5x_{soja} + 0.4x_{peixe} &\geq 0.3 \quad (*0.325) \\
0.065x_{osso} + 0.1625x_{soja} + 0.13x_{peixe} &\geq 0.0975 \\
0.6x_{osso} + 0.4x_{soja} + 0.4x_{peixe} &\geq 0.5 \quad (*0.825) \\
0.495x_{osso} + 0.495x_{soja} + 0.33x_{peixe} &\geq 0.4125
\end{aligned}
$$

$$
\begin{aligned}
\text{minimizar} \quad & 0.56x_{osso} + 0.81x_{soja} + 0.46x_{peixe}, \\
\text{sujeito a} \quad & 0.2x_{osso} + 0.5x_{soja} + 0.4x_{peixe} \geq 0.3, \\
& 0.6x_{osso} + 0.4x_{soja} + 0.4x_{peixe} \geq 0.5, \\
& x_{osso} + x_{soja} + x_{peixe} = 1, \\
& x_{osso} \geq 0, \quad x_{soja} \geq 0, \quad x_{peixe} \geq 0.
\end{aligned}
$$

$$
\begin{aligned}
0.065x_{osso} + 0.1625x_{soja} + 0.13x_{peixe} &\geq 0.0975 \\
0.495x_{osso} + 0.33x_{soja} + 0.33x_{peixe} &\geq 0.4125
\end{aligned}
$$

somando

$$
0.56x_{osso} + 0.4925x_{soja} + 0.46x_{peixe} \geq 0.51
$$

Custo:

$$
0.56x_{osso}+0.81x_{soja}+0.46x_{peixe} \geq 0.56x_{osso}+0.4925x_{soja}+0.46x_{peixe} \geq 0.51
$$

Logo o custo obrigatoriamente será maior ou igual a 0.51. Se já temos uma solução de custo 0.51 ela obviamente é ótima.

OR-Tools

- É uma ferramenta de código aberto para otimização combinatória.
- Em geral, o objetivo é encontrar a melhor solução para um problema em um conjunto muito grande de soluções possíveis.
- Exemplos de problemas que podem ser resolvidos com o OR-Tools:
  - Roteamento de Veículos: Encontrar as melhores rotas para uma frota de veículos que coletam e entregam pacotes dadas algumas restrições ("um caminhão não pode carregar mais do que 23 toneladas", "todas as entregas devem ser feitas dentro de uma janela de 8 horas", etc)
  - Escalonamento: Encontrar o escalonamento (agendamento) ótimo para um conjunto complexo de tarefas, sujeito a algumas restrições ("algumas atividades precisam ser executadas antes de outras atividades", "algumas atividades podem compartilhar um mesmo recurso", etc)
  - *Bin packing*: Empacotar vários objetos pequenos em recipientes, minimizando o número de recipientes necessários.

OR-Tools

OR-Tools inclui resolvedores:

- Programação por Restrições.
- Programação Linear.
- Programação Linear Inteira Mista.
- Roteamento de Veículos.
- Algoritmos em Grafos (caminho mais curto, fluxo de custo mínimo, fluxo máximo, etc)

OR-Tools

OR-Tools é escrito em **C++**, mas também pode ser usado com Python, Java ou C#.

- Link para instalação:
  `https://developers.google.com/optimization/install/cpp`
- Meu sistema é o Linux Mint 19.3 Tricia, então baixei a versão do OR-Tools para Ubuntu 18.04
- Link para vários exemplos:
  `https://developers.google.com/optimization/examples`

OR-Tools

$$
\begin{aligned}
\text{maximize} \quad & 3x + y, \\
\text{sujeito a} \quad & x + y \leq 2, \\
& 0 \leq x \leq 1, \\
& 0 \leq y \leq 2.
\end{aligned}
$$

$$\begin{aligned} \text{maximize} \quad & 3x + y, \\ \text{sujeito a} \quad & x + y \leq 2, \\ & 0 \leq x \leq 1, \\ & 0 \leq y \leq 2. \end{aligned}$$

```
#include "ortools/linear_solver/linear_solver.h"
using namespace operations_research;
int main() {
  // Criar o Resolvedor, que usa o GLOP
  MPSolver solver("simple_lp_program",
                  MPSolver::GLOP_LINEAR_PROGRAMMING);

  // Criar as variaveis x e y, ja dizendo o tipo,
  // os limitantes, e um nome
  MPVariable* const x = solver.MakeNumVar(0.0, 1, "x");
  MPVariable* const y = solver.MakeNumVar(0.0, 2, "y");

  // Criar a restricao, 0 <= x + y <= 2.
  MPConstraint* const ct = solver.MakeRowConstraint(0.0, 2.0,
                                                    "ct");
  ct->SetCoefficient(x, 1);
  ct->SetCoefficient(y, 1);
```

$$\begin{aligned} \text{maximize} \quad & 3x + y, \\ \text{sujeito a} \quad & x + y \leq 2, \\ & 0 \leq x \leq 1, \\ & 0 \leq y \leq 2. \end{aligned}$$

```
  // Criar a funcao objetivo, 3 * x + y.
  MPObjective* const objective = solver.MutableObjective();
  objective->SetCoefficient(x, 3);
  objective->SetCoefficient(y, 1);
  objective->SetMaximization();

  // Resolver!
  solver.Solve();

  std::cout << "Solution:" << std::endl;
  std::cout << "Objective value = " << objective->Value()
                                    << std::endl;
  std::cout << "x = " << x->solution_value() << std::endl;
  std::cout << "y = " << y->solution_value() << std::endl;
  return EXIT_SUCCESS;
}
```

```
hokama@fanatico:~$ cd /opt/or-tools_Ubuntu-18.04-64bit_v7.6.7691/
hokama@fanatico:/opt/or-tools_Ubuntu-18.04-64bit_v7.6.7691$ make run SOURCE=/home/hokama/cic111/teste.cc
g++ -fPIC -std=c++11 -O4 -DNDEBUG -Iinclude -I. -DARCH_K8 -Wno-deprecated -DUSE_CBC -DUSE_CLP -DUSE_BOP -DUSE_GL
OP \
 objs/teste..o \
 -Llib -Llib64 -lprotobuf -lglog -lgflags -lCbcSolver -lCbc -lOsiCbc -lCgl -lClpSolver -lClp -lOsiClp -lOsi -lCo
inUtils -lortools -Wl,-rpath,"\$ORIGIN" -Wl,-rpath,"\$ORIGIN/../lib64" -Wl,-rpath,"\$ORIGIN/../lib" -lz -lrt -lp
thread \
 -o bin/teste
bin/teste
Solution:
Objective value = 4
x = 1
y = 1
hokama@fanatico:/opt/or-tools_Ubuntu-18.04-64bit_v7.6.7691$
```

- Uma construtora lucra 4.6 milhões com a venda de um prédio comercial e 2.2 milhão com a venda de casas de luxo.
- Para construir um prédio comercial a construtora aloca 40 funcionários, e para as casas de luxo, 19 funcionários. Sendo que a construtora tem um total de 100 funcionários. As duas obras gastam o mesmo tempo.
- Em quantos prédio comerciais e casas de luxo a construtora deve investir?

x é o número de prédios.
y é o número de casas.

$$\begin{aligned} \text{maximizar} \quad & 4.6x + 2.2y, \\ \text{sujeito a} \quad & 40x + 19y \leq 100, \\ & x \geq 0, \quad y \geq 0. \end{aligned}$$

$$\begin{aligned} \text{maximizar} \quad & 4.6x + 2.2y, \\ \text{sujeito a} \quad & 40x + 19y \leq 100, \\ & x \geq 0, \quad y \geq 0. \end{aligned}$$

- A solução ótima para esse modelo é $x = 0$ e $y = 5.26$ que dá um lucro de 11.58 milhões.
- Infelizmente construir 26% de uma casa não dá lucro algum, de fato pode até trazer algum prejuízo.
- Podemos arredondar para baixo (5 casas) porém essa solução não é mais ótima.
- Precisamos que as variáveis sejam inteiras!

$$\begin{aligned} \text{maximizar} \quad & 4.6x + 2.2y, \\ \text{sujeito a} \quad & 40x + 19y \leq 100, \\ & x \geq 0, \quad y \geq 0. \\ & x, y \in \mathbb{Z} \end{aligned}$$

- A solução ótima desse modelo é $x = 2$ e $y = 1$.

Programação Linear Inteira (PLI)

$$\begin{aligned} \text{maximizar} \quad & 4.6x + 2.2y, \\ \text{sujeito a} \quad & 40x + 19y \leq 100, \\ & x \geq 0, \quad y \geq 0. \\ & x, y \in \mathbb{Z} \end{aligned}$$

- Quando tempos variáveis inteiras, utilizamos a Programação Linear Inteira - PLI (Integer Linear Programming - ILP).
- Quando temos algumas variáveis inteiras e outras continuas temos a Programação Linear Mista (Mixed-Integer Programming - MIP).
- Infelizmente resolver um ILP ou MIP é NP-Difícil.

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.

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

Limitantes

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

Capacidade da Mochila: 14kg

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 $z^* \geq 34$

Limitantes

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

Limitante Superior:

- 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.

Limitantes Superior - Relaxação

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 $z^* \leq 40$

Limitantes Superior - Relaxação

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

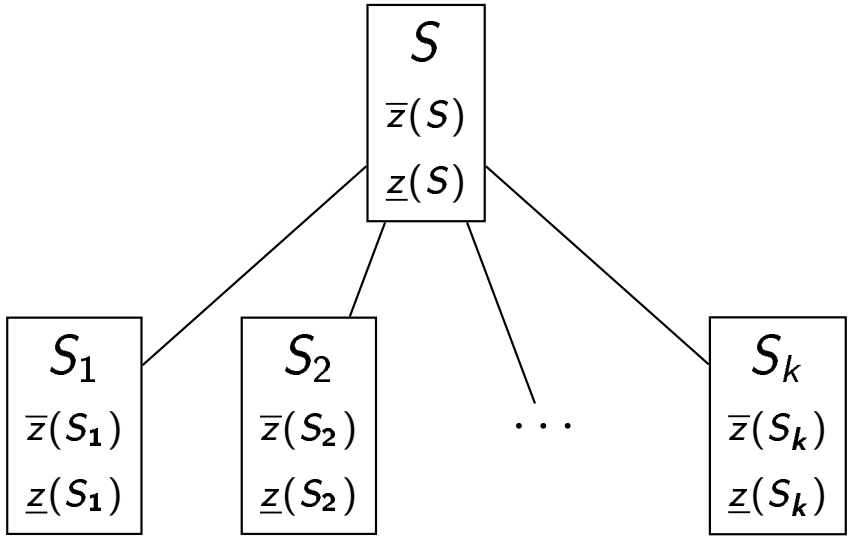
Pesos	**2**	5	4	**6**	**5kg**
Valor	**10**	11	10	**13**	**14**

Solução: 37.

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 $f : S \rightarrow \mathbb{R}$ 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.

Branch-and-Bound



- 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**.

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.

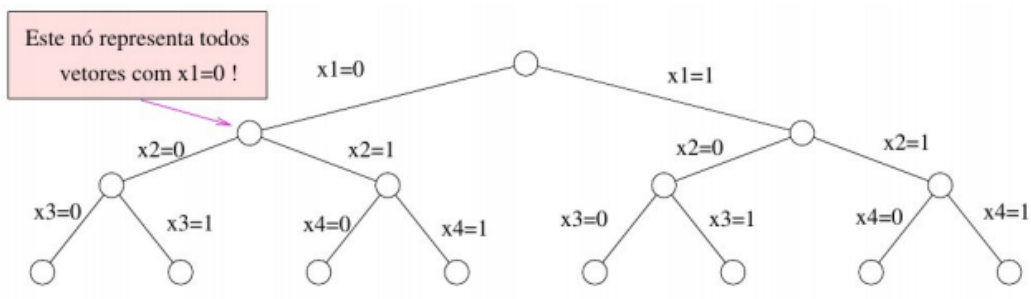


Figura 1.8. Árvore de enumeração completa para o problema binário da mochila com 3 itens.

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

Ramificações

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