Universidade Estadual de Campinas
Instituto de Matemática, Estatística e Computação Cientifica

Problema da Mochila

Disciplina MS777 - Projeto Supervisionado I
Autor: Rubens Carvalho - ra: 122181
Orientador: Prof[a] Dr. Kelly Cristina Poldi

Campinas - SP
junho de 2015.

Resumo

Neste projeto, estudamos o problema da mochila e suas variações. Este problema é um dos mais conhecidos na literatura e facilmente encontrado em situações reais, como investir um capital em várias aplicações por exemplo. Devido a sua importância teórica, e por estar na classe dos problemas NP-hard, diversos autores buscam formas mais eficientes de resolvê-lo se aproveitando de características próprias de cada variação do problema. Apresentamos um resumo dos modelos mais conhecidos dentre todos os existentes, suas características, a diferença entre eles e como são modelados matematicamente. Juntamente com a modelagem, apresentamos dois métodos de solução efetivos para os problemas da mochila limitada e da mochila binária, o algoritmo utilizado e os resultados obtidos computacionalmente.

Palavra-chave: Problema da Mochila; Programação Dinâmica; Enumeração Implícita.

Abstract

In this project, we study the knapsack problem and variations. This problem is one of the most known in the literature and easily found in real situations such as investing capital in various applications for example. Due to its theoretical importance, and to be in the class of NP-hard problems, several authors seek more efficient ways to solve it taking advantage of the characteristics of each variation of the problem. A summary of the best known of all the existing models, their characteristics, the difference between them and how they are modeled mathematically. Along with the modeling, we present two methods effective solution to the problems of bounded knapsack problem and 0-1 knapsack problem, the algorithm used and the results obtained computationally.
Keywords: Knapsack problem; Dynamic programming; Branch and bound

1 Introdução

Imagine que você irá realizar uma viagem de alguns dias, obviamente necessitará levar uma mochila com alguns itens dentro. Se você for uma pessoa organizada provavelmente fará uma pequena lista com todos os objetos que poderia levar (roupas, livros, aparelhos eletrônicos, etc.), mesmo que essa lista seja imaginária ela irá existir em algum momento.

No entanto, uma mochila não é infinita e cada item tem um peso, por exemplo uma mochila que comporta até quatro livros não consegue levar cinco ou mais livros de tamanhos semelhantes na viagem. Logo, muitas vezes algo pode ser deixado para trás. Essa ideia gera uma pergunta bastante simples, mas com uma resposta nem tão simples as vezes: o que devo levar?

Estamos na seguinte situação: você possui apenas uma mochila para a viagem e vários itens para levar, mas não cabem todos na mochila. Uma maneira de resolver isso é atribuir valor de importância para cada objeto e levar na viagem apenas os mais importantes, coloco na mochila o item mais importante dentre todos e, em seguida, o segundo item mais importante que caiba no espaço restante e assim sucessivamente até completar a mala. Esse método se chama "Método Guloso", é simples e rápido, mas nem sempre trás bons resultados (imagine que você preencheu a mochila com um item muito grande e não sobrou espaço para nenhum outro item, por exemplo roupas).

Assim é definido o problema da mochila, temos um número limitado de itens para colocar em uma mochila levando em consideração o peso de cada item e a capacidade do recipiente, e queremos maximizar o valor total de utilidade, sendo que a cada item é associado o seu valor de utilidade. Na próxima seção apresentamos a formulação matemática desse problema e de algumas variações dele.

O problema da mochila é um problema de programação linear inteira e é classificado como NP-hard, caso alguém deseje listar todas as possibilidades e tomar a maior delas, a complexidade do problema cresce na ordem de 2^n, onde n é o número de itens. Supondo um computador que analisa 1 bilhão de vetores por segundo e retorna a melhor combinação, iríamos precisar de aproximadamente 30 anos para analisar um caso com 60 itens, mais de 60 anos para um caso com 61 itens, conforme Martello e Toth (1990) apresentaram. Porém, alguns algoritmos precisam de apenas poucos segundos para resolver problemas com 1000000 itens em um computador simples.

A modelagem matemática será discutida na seção 2 e os métodos de solução serão apresentados na seção 3.

2 O Problema da Mochila

Apresentamos a seguir algumas formulações para o problema da mochila (*Knapsack Problem*) mantendo uma visão próxima aos estudos de Martello e Toth (1990) por serem um dos principais pesquisadores nessa área.

2.1 Mochila 0 - 1

O problema da mochila 0 - 1, ou mochila binária, consiste em, dada uma mochila de capacidade L e m itens cujos peso p_i e valor de utilidade v_i são dados, escolher quais itens serão alocados na mochila. Considere a variável de decisão x_j de maneira que, se o item j é alocado na mochila, então $x_j = 1$, caso contrário, então $x_j = 0$. Assim, o modelo matemático para o problema da mochila 0-1 é dado por,

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} v_i x_i \\ \text{s.a} \quad & \sum_{i=1}^{m} p_i x_i \leqslant L \\ & x_i \in \{0,1\}, i = 1, \ldots, m \end{aligned}$$

Esta variação é uma das mais estudadas dentre os problemas de mochila pois pode ser visto como um simples problema de programação inteira, ele aparece como subproblema em muitos problema mais complexos e pode representar muitas situações práticas do dia-a-dia.

Para ilustrar, considere o seguinte problema com 5 itens e uma mochila de capacidade 10:

$$\begin{aligned} \max \quad & z = 4x_1 + 6x_2 + 5x_3 + 3x_4 + x_5 \\ \text{s.a} \quad & 5x_1 + 4x_2 + 3x_3 + 2x_4 + x_5 \leqslant 10 \\ & x_i \in \{0,1\}, i = 1, \ldots, 5 \end{aligned}$$

Neste exemplo temos uma situação com 5 itens no total, cada um com seu respectivo peso (p_i, i = 1,...,5) e valor de utilidade (v_i, i = 1,...,5), como visto na tabela abaixo:

Tabela 1: Dados relacionados ao exemplo 1 no problema da mochila.

	x_1	x_2	x_3	x_4	x_5
v_i	4	6	5	3	1
p_i	5	4	3	2	1

Um exemplo simples com solução ótima dada pelo vetor $x = (0, 1, 1, 1, 1)$, os métodos de solução serão discutidos na seção 3 . Um exemplo mais prático seria onde um gerente de uma empresa possui um capital L e deseja investir em alguns projetos dentre uma lista com m opções. Cada projeto possui um custo (p_i) para ser desenvolvido e um lucro de retorno (v_i), que pode ser desde redução de custo ou aumento de produtividade.

2.2 Mochila Inteira

Esta é uma variação mais genérica do problema anterior da mochila 0-1. Anteriormente, tínhamos apenas a opção de colocar, ou não, o item no recipiente, agora existe uma quantidade infinita do mesmo item para ser alocada. Podemos ver sua modelagem a seguir:

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} v_i x_i \\ \text{s.a} \quad & \sum_{i=1}^{m} p_i x_i \leqslant L \\ & x_i \in \mathbb{N}; i = 1, \ldots, m \end{aligned}$$

onde,

- $x_i \in \mathbb{N}$: quantidade do item i colocada na mochila.

- v_i: valor de utilidade do item i.
- p_i: peso do item i.
- L: capacidade da mochila.
- m: número de tipos de itens.

É fácil notar a semelhança deste com o problema anterior. A diferença mais marcante é que a variável de decisão x_i não esta mais sujeita ao conjunto binário $\{0, 1\}$, ela é livre para tomar qualquer valor dentro de $\mathbb{N}$, incluindo o 0 caso o item não seja alocado.

2.3 Mochila Limitada

Em problemas reais, nem sempre temos apenas um item de cada tipo no problema, e muito menos temos infinitos itens a nossa disposição. Normalmente dispomos de um intervalo fechado $[0, d_i]$ para cada variável $x_i \in \mathbb{N}$, onde $d_i \in \mathbb{N}$ é a quantidade máxima de itens de peso p_i que podem ser alocados na mochila. A formulação, neste caso, é a seguinte:

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} v_i x_i \\ \text{s.a} \quad & \sum_{i=1}^{m} p_i x_i \leqslant L \\ & 0 \leqslant x_i \leqslant d_i, i = 1, \ldots, m \\ & x_i \in \mathbb{N} \end{aligned}$$

Iremos mostrar um exemplo para esse tipo de problema da mochila na seção 4.

2.4 Mochila Compartimentada

Podemos considerar o caso no qual os itens a serem alocados em uma mochila estejam separados em classes distintas,(por exemplo, classe 1: comida; classe 2: roupas; classe 3: livros; etc.) e cada item deve ser agrupado com os demais itens de sua classe dentro da mochila, ou seja, cada compartimento da mochila deve ter apenas itens da mesma classe. Cada compartimento é flexível, uma pessoa pode levar mais peso em livros do que de comida, entretanto, a capacidade de cada compartimento é limitada inferiormente e superiormente.

Um custo c_k é associado a cada compartimento caso ele seja usado com algum item da classe k e o somatório de todos os compartimentos não pode exceder a capacidade total da mochila. O problema da mochila compartimentada consiste em determinar as capacidades adequadas de cada compartimento e como estes devem ser carregados de modo que o valor de utilidade total seja máximo. Hoto (1996) fornece os primeiros passos para o problema da mochila compartimentada e métodos heurísticos para obtenção de solução. Mais tarde, Hoto (2001), apresenta métodos exatos e heurísticos para a mochila compartimentada.

Uma boa definição deste problema é dada por Marques e Arenales (2002): "O Problema da Mochila Compartimentada consiste em determinar as capacidades adequadas de cada compartimento e como estes devem ser carregados de modo que o valor de utilidade total (soma dos valores de utilidade de todos os itens selecionados) seja máximo, descontando-se os custos dos compartimentos, os quais dependem dos agrupamentos com que foram preenchidos (c_k)"

Por ser um problema mais complexo, tanto para se enunciar como para resolver, foge do intuito desde trabalho. Para mais detalhes, veja Hoto (1996). Hoto (2001) e Marques e Arenales (2002).

2.5 Múltiplas Mochilas

Até o momento, vimos problemas onde queríamos completar apenas uma mochila, mas existem problemas onde queremos carregar várias mochilas. Para cada problema descrito acima existe um semelhante envolvendo esta ideia de múltiplas mochilas. Martello e Toth (1990) fizeram uma modelagem matemática para o problema de múltiplas mochilas 0-1.

$$\begin{aligned} \max \quad & \sum_{j=1}^{n}\sum_{i=1}^{m} v_i x_{ij} \\ \text{s.a} \quad & \sum_{i=1}^{m} p_i x_{ij} \leqslant L_j \ , \ j = 1, \ldots, n \\ & \textstyle\sum_{j=1}^{n} x_{ij} \leqslant 1 \ , \ i = 1, \ldots, m \\ & x_{ij} \in \{0,1\}, i = 1, \ldots, m \ e \ j = 1, \ldots, n \end{aligned}$$

onde,

- x_{ij}: informa se o item i está ($x_{ij} = 1$), ou não ($x_{ij} = 0$), na mochila j.
- L_j: capacidade da mochila j.
- m: npumero de itens.
- n: número de mochilas.

Se $n = 1$ temos apenas 1 mochila envolvida no problema e a modelagem acima se torna idêntica ao problema apresentado na subseção 2.1.

Vimos até agora os problemas clássicos e mais comuns dentro do problema da mochila, ainda existem outros casos menos trabalhados, como o problema do troco, onde o peso carregado na mochila deve ser igual ao seu limite máximo e o problema da mochila fracionada, onde cada item pode ser dividido em itens menores e, consequentemente, com valor de importância proporcional. A figura (1) é um exemplo ilustrando o caso de uma mochila binária fracionada. Na parte **a)** é apresentado os itens com seus respectivos pesos e valores e a capacidade total da mochila; em **b)** temos possíveis soluções para o caso binário, onde a primeira opção seria a solução ótima encontrada pela programação dinâmica e a segunda seria a solução obtida pelo método guloso levando em consideração a razão entre o peso e o valor de cada item; em **c)** temos a mochila fracionada, onde colocamos os itens 1 e 2 e uma fração do 3º item para completar a mochila e não deixar sobra.

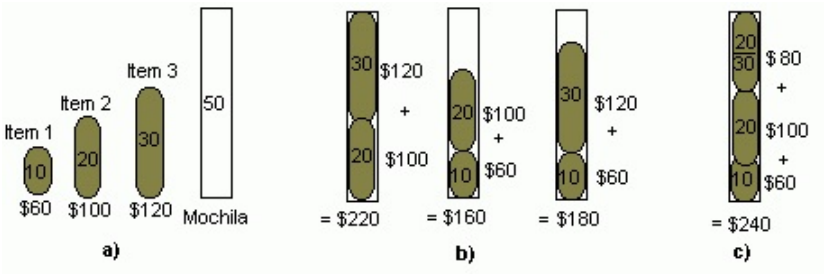

Figura 1: Representação gráfica de uma mochila fracionada (fonte; www.google.com.br).

3 Métodos de solução

Nesta seção, iremos apresentar dois métodos de solução para o problema da mochila. Apresentamos o método de programação dinâmica para o problema da mochila 0-1. A seguir, apresentamos o método de enumeração implícita, ou método de *Branch-and-Bound*, para o problema da mochila inteira.

3.1 Programação Dinâmica

Cada instância do problema é resolvida a partir da solução de instâncias menores. A característica distintiva da programação dinâmica é a tabela que armazena as soluções das várias instâncias. É um método que resolve várias mochilas menores dentro do problema original até atingir o tamanho máximo definido, explicando de uma maneira mais sucinta, dada uma mochila de tamanho L e os itens x_i com seus respectivos pesos, a programação dinâmica resolve recursivamente o problema para uma mochila de tamanho 0. Em seguida resolve o problema para uma mochila de tamanho 1, depois para uma mochila de tamanho 2 e assim sucessivamente até obter uma solução para a mochila de tamanho L.

A programação dinâmica é um procedimento de otimização aplicável a problemas solucionáveis usando-se esta sequência de decisões, por sua vez, resultando numa sequência de estados ou caminho e, num valor associado a cada caminho que deve ser minimizado ou maximizado. Duas características são comuns nestes procedimentos:

i) um dado problema pode ser resolvido se as melhores soluções para certos estados deste problema forem determinadas;

ii) os estados finais deste problema devem ser tão simples a ponto de terem soluções triviais.

Considere o seguinte problema da mochila binária,

$$\begin{aligned} \text{g(x)} = \max \quad & v_1x_1 + v_2x_2 + \ldots v_mx_m \\ \text{s.a} \quad & p_1x_1 + p_2x_2 + \ldots p_mx_m \leqslant L \\ & x_i \in \{0,1\}, i = 1, \ldots, m \end{aligned}$$

Um método de programação dinâmica para a resolução do problema da mochila consiste numa fórmula recursiva utilizando uma estratégia de percorrer todas as possíveis decisões, estudado por Yanasse e Soma (1987). A solução de um nó representando um espaço livre X, cujo valor da função objetivo é denotado por $G(i, X)$. Vamos denotar $G(i, X)$ como o valor máximo obtido ao combinar os valores de peso $p_1, p_2, \ldots, p_i, 1 \leqslant i \leqslant m$ ao longo do compartimento $X = 0, 1, \ldots, L$. Ou seja,

$$\begin{aligned} \text{G(i,X)} = \max \quad & \sum_{k=1}^{i} v_kx_k \\ \text{s.a} \quad & \sum_{k=1}^{i} p_kx_k \leqslant X \\ & x_k \in \{0,1\}, k = 1, \ldots, i \end{aligned}$$

Os valores X = 0, 1,..., L são chamados estados e representam diferentes configurações do problema durante o processo de decisão. A fórmula de recorrência que define a solução de cada estado é dada por:

$$G(i,X) = \begin{cases} G(i-1,X) & se \;\; p_i > X \\ max\ \{G(i-1,X), G(i-1,X-p_i)+v_i\} & se \;\; p_i \leqslant X \end{cases}$$

para $1 \leqslant i \leqslant m, 0 \leqslant X \leqslant L$.

Temos que G(i,X) é igual a $G(i-1,X)$ caso o novo item é maior do que a capacidade atual da mochila, caso contrário ele será igual ao máximo entre o estado atual sem o item ($G(i-1,X)$) e $G(i-1,X-p_i)+v_i$ onde é colocado o item, a capacidade é reduzida de acordo com o peso do item colocado e seu valor é acrescentado na função.

$G(m,X)$ é o valor máximo obtido para o estado X e depende de uma escolha de i, o que corresponde a um estágio no processo de decisão. Este valor é o resultado da sequência de decisões (multiestágio) tomadas até o estágio X. $G(m,L)$ corresponde à solução ótima do problema da mochila. Abaixo é apresentado o pseudocódigo deste método:

```
Início Procedimento  PDinamica (v, p, m, L)
    Para j = 0 ate L faça
        G(0,j) = 0;
    fim para
    Para i = 1 ate m faça
        Para j = 0 ate L faça
            se ((p(i) <= j) e (v(i) + G(i - 1,j - p(i)) > G(i - 1,j)))  então
                G(i,j) = v(i) + G(i - 1,j - p(i));
                dec(i,j) = 1;
            senão
                G(i,j) = G(i - 1,j);
                dec(i,j) = 0;
            fim se
        fim para
    fim para
    Para i = m ate 1 faça
        se (dec(i,j) == 1)
            exiba i;
            j = j-p(i);
         fim se
    fim para
    Retorne G(m,L);
Fim Procedimento
```

O vetor $dec(i,j)$ guarda quais itens foram colocados na mochila e, ao final do algoritmo, ele é percorrido para determinar a solução ótima. Na seção 4.1 apresentamos um exemplo desde algoritmo e os resultados obtidos.

3.2 Enumeração Implícita

O seguinte método da enumeração implícita foi proposto por Gilmore e Gomory (1963), para o problema da mochila. Este método é implementado utilizando uma busca em profundidade primeiro e consiste em enumerar implicitamente todas as soluções do problema da mochila. Para isto, utiliza-se limitantes para o problema que permitem eliminar soluções cujos resultados são inferiores que a atual, sem perder a otimalidade. Este método é importante, pois gerar

explicitamente todas as possíveis soluções do problema da mochila, para exemplos relativamente grandes, se torna muito caro computacionalmente.

Neste métodos, iremos resolver problemas onde a quantidade dos itens a serem escolhidos é limitada dentro de um intervalo do tipo $[0, d_i]$, onde $d_i \in \mathbb{N}$ é a quantidade máxima disponível do item x_i que pode ser alocado na mochila. Por convenção iremos utilizar 0 como limite inferior neste método, ou seja, o item x_i pode ser colocado até d_i vezes ou não ser colocado. É fácil notar que se mudarmos o limite inferior, digamos de 0 para 5, significa que o item deve ser colocado na mochila obrigatoriamente com, no mínimo, 5 unidades. Basta então adicionar essa quantidade na mochila, retirar seu peso da capacidade total L e retornamos novamente com limite inferior sendo 0 como foi estabelecido anteriormente.

A seguir, apresentaremos um algoritmo de enumeração para resolução do seguinte problema da mochila:

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} v_i x_i \\ \text{s.a} \quad & \sum_{i=1}^{m} p_i x_i \leqslant L \\ & 0 \leqslant x_i \leqslant d_i, i = 1, \ldots, m \\ & x_i \in \mathbb{N} \end{aligned}$$

Inicio Procedimento Enumeração Implícita
Dados os vetores v, p, d e os inteiros m e L,

Passo 1: % Definição das variáveis mais valiosas
Faça $\theta_i = \frac{v_i}{p_i}$, com $i = 1, 2, \ldots, m$, e reordene as variáveis de modo que $\theta_1 \geqslant \theta_2 \geqslant \cdots \geqslant \theta_m$
Faça $v_{m+1} = 0$, $p_{m+1} = 1$, $\underline{G} = 0$ e $g(\sigma) = 0$

Passo 2: % Definição da solução inicial
Determine $\sigma = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ de modo

$$\alpha_1 = min\left\{ \left\lfloor \frac{L}{p_1} \right\rfloor, d_1 \right\} \Rightarrow resta\ L - p_1\alpha_1$$

$$\alpha_2 = min\left\{ \left\lfloor \frac{L - p_1\alpha_1}{p_2} \right\rfloor, d_2 \right\} \Rightarrow resta\ L - p_1\alpha_1 - p_2\alpha_2$$

$$\vdots$$

$$\alpha_m = min\left\{ \left\lfloor \frac{L - \sum_{j=1}^{m-1} p_j\alpha_j}{p_m} \right\rfloor, d_m \right\}$$

Passo 3: % Avaliação da solução obtida
Faça $g(\sigma) = \sum_{i=1}^{m} v_i\alpha_i$
Se $\underline{G} < g(\sigma)$ então $\underline{G} = g(\sigma)$ e $\sigma^* = \sigma$

Passo 4: % Teste de otimalidade e limitante superior

Determine o maior índice k tal que $\alpha_k \neq 0$

Se $\sigma = 0$ então PARE, σ^* é a melhor solução.

Caso contrário, calcule

$$\overline{G}(\sigma) = v_1\alpha_1 + \cdots + v_k(\alpha_k - 1) + \frac{v_{k+1}}{p_{k+1}}(L - p_1\alpha_1 - \cdots - p_k(\alpha_k - 1))$$

Passo 5: % Backtracking

Se $\overline{G}(\sigma) \leqslant \underline{G}$ então faça $\alpha_k = 0$ e volte ao *Passo 4*.

Caso contrário, faça $\alpha_k = \alpha_k - 1$ e defina σ com

$$\alpha_j = min\left\{\left\lfloor \frac{L - \sum_{i=1}^{j-1} p_i\alpha_i}{p_j}, d_j \right\rfloor\right\}, j = k+1, \ldots, m$$

e volte ao *Passo 3*.

Fim Procedimento.

Ao final do método, $g(\sigma)$ guarda o valor ótimo encontrado e o vetor σ é a solução do problema.

Para exemplificar este método escrevemos o algoritmo em linguagem C e apresentamos os resultados na seção 4.2.

4 Implementação Computacional

Os algoritmos de Programação Dinâmica e Enumeração Implícita foram implementados em linguagem C e executados em um notebook com 4 Gb de memória RAM ddr3, processador *intel*® i5 2.50 GHz e S.O. Windows 8.1 64bits.

Alguns exemplos foram executados durante os testes. Apresentamos, a seguir, um exemplo para cada algoritmo estudado para exemplificar os métodos.

4.1 Programação Dinâmica

Considere uma mochila de capacidade L = 10 e 5 itens com seus pesos e valores dados pela tabela a seguir,

Tabela 2: Dados de peso e capacidade relativos aos itens do exemplo 2

Itens	1	2	3	4	5
Valor	100	60	70	15	15
Peso	8	3	6	4	2

Através do algoritmo apresentado na seção 3.1, obtemos recursivamente a melhor solução para todas as capacidades de mochila até L = 10. O valor ótimo encontrado é de $G(m, L) = G(5, 10) = 130$ cujo vetor solução é $\alpha = (0, 1, 1, 0, 0)^t$. Todas as soluções obtidas pelo programa podem ser vistas na tabela a seguir.

Tabela 3: Valores obtidos na solução do exemplo 2

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1(8)	0	0	0	0	0	0	0	0	100	100	100
2(3)	0	0	0	60	60	60	60	60	100	100	100
3(6)	0	0	0	60	60	60	70	70	100	130	130
4(4)	0	0	0	60	60	60	70	75	100	130	130
5(2)	0	0	15	60	60	75	75	75	100	130	130

4.2 Enumeração Implícita

Vamos utilizar uma mochila de capacidade $L = 629$ para alocar 30 itens, cujos pesos e importâncias estão exibidos nos vetores abaixo:

$p_i = (82, 26, 42, 36, 70, 77, 36, 77, 55, 10, 96, 23, 2, 82, 14, 22, 50, 69, 39, 71, 44, 96, -2, 51, 65, 9, 50, 91, 65, 25)$
$v_i = (92, 29, 37, 37, 77, 83, 28, 83, 52, 5, 106, 31, -4, 89, 14, 21, 44, 90, 45, 78, 31, 101, 5, 40, 66, 11, 59, 88, 73, 29)$
$d_i = (4, 5, 3, 6, 2, 7, 1, 8, 1, 4, 1, 3, 5, 6, 3, 2, 3, 4, 5, 6, 4, 5, 6, 1, 9, 8, 7, 4, 2, 5)$

Com tais valores o método obteve o valor de $g(\sigma) = 777$ com o seguinte vetor solução:

$\sigma_i = (0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 3, 1, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 8, 3, 0, 0, 2)$

Por serem exemplos relativamente simples, o tempo de execução, tanto para a programação dinâmica quanto para o método de Branch and bound, é inferior ao tempo da máquina.

5 Conclusão

Estudamos os modelos de mochila mais conhecidos apresentando suas principais características e suas modelagem. É fácil notar que este problema pode ser aplicado em diversas áreas para modelar problemas reais e encontrar uma solução ótima, ou uma aproximação factível em alguns casos, utilizando métodos computacionais desenvolvidos anteriormente, como a programação dinâmica ou heurísticas próprias. Durante a implementação computacional alguns aspectos podem ser destacados: ao final do processo da programação dinâmica, conhecemos os valores ótimos de todas as mochilas de tamanho menor a mochila estudada. Isso é importante pois em casos onde se deseja conhecer várias soluções para diferentes mochilas, basta resolver o problema para a mochila de maior capacidade e, como consequência, irá obter a solução de todas as mochilas de capacidade inferior sem necessidade de resolver outro problema, no entanto, se torna um problema se o objetivo é obter uma solução sem utilizar muitos recursos do computador pois este método aloca um vetor do tamanho da mochila na memória. Diferentemente da enumeração implícita, que resolve apenas uma mochila de cada vez, mas requer menos da máquina para resolver o problema.

Bibliografia

- Gilmore, P. C., Gomory, R. E. (1963). "A linear programming approach to the cutting stock problem - Part II". Operations Research, 11(6):863-888.
- Hoto, R.S.V. (1996). "Otimização no Corte de Peças Unidimensionais com Restrições de Agrupamento". Dissertação de Mestrado, ICMC-USP, São Carlos, São Paulo.
- Hoto, R.S.V. (2001). "O Problema da Mochila Compartimentada Aplicado no Corte de Bobinas de Aço". Tese de Doutorado, COPPE-UFRJ, Rio de Janeiro, Rio de Janeiro.
- Martello, S.; Toth, P. (1990). "Knapsack Problems: Algorithms and Computer Implementations."John Wiley & Sons, Chichester.
- Marques F.P.; Arenales M. N. (2002). "O problema da mochila compartimentada e aplicações". Pesquisa Operacional, 22(3):285-304
- Poldi, K. C. (2003). "Algumas extensões do problema de corte de estoque". Dissertação de Mestrado. ICMC-USP, São Carlos, São Paulo.
- Yanasse, H.H.; Soma, N.Y. (1987). "A new enumeration scheme for the knapsack problem". Discrete Applied Mathematics, 18:235-245.