Problema da Mochila com Conflitos: resolução usando Branch-and-Bound e busca ordenada por densidade

Daniela C.Terra[1], Haroldo Gambini Sandos[1], Rodrigo Cesar Pedrosa Silva

[1]Programa de Pós-Graduação em Ciência da Computação
Departamento de Computação (DECOM)
Instituto de Ciências Exatas e Biológicas (ICEB)
Universidade Federal de Ouro Preto (UFOP)
Ouro Preto – MG – Brasil

daniela.terra@ifmg.edu.br, haroldo@ufop.edu.br, rodrigo.silva@ufop.edu.br

Resumo. *Este trabalho implementa uma proposta de solução para o problema de otimização conhecido como problema da mochila com conflitos. A solução proposta baseou-se na busca por soluções usando a técnica denominada branch-and-bound. O programa foi escrito em Java e testado para 20 instâncias de tamanhos variando entre 50 à 1000 itens. A heurística utilizada ordenou o espaço de buscas por densidade de lucro. Nesse caso o limite superior de cada nó foi calculado com base na maior densidade dentre os itens restantes. O algoritmo encontrou soluções viáveis para todas as instâncias, sem exceder o tempo limite da máquina utilizada.*
PALAVRAS-CHAVE: otimização combinatória, Knapsack Problem, branch-and-bound

1. Introdução

O problema da mochila (PM) é problema de otimização combinatória onde dado um conjunto de N itens, uma capacidade máxima (c), uma lista de itens e respectivos valores (lucros) busca-se a melhor solução em termos do conjunto de itens de maior lucro que não exceda a capacidade C. No caso da versão com conflitos, as restrições são para itens conflitantes os quais não podem estar juntos na mesma solução.

A abordagem de busca exaustiva para este problema levaria à geração de todos os subconjuntos de um conjunto de n itens dados, para encontrar aquele de maior valor entre eles. Como o número de subconjuntos existentes é 2^n o custo da busca exaustiva é $\Omega(2^n)$, não importando quão eficientemente os subconjuntos individuais sejam gerados. As soluções baseadas nesta enumeração de subconjuntos produz algoritmos ineficientes para a maioria das entradas. O problema da mochila é da classe dos problemas do tipo NP-difíceis para os quais nenhum algoritmo de tempo polinomial é conhecido [Cormen et al. 2009].

Para dar mais inteligência à busca o branch-and-bound pode ser usado. Essa técnica é apropriado para problemas de otimização em que a busca deve maximizar ou minimizar uma função objetivo, obedecendo à restrições adicionais se existirem. A Figura 1 exibe a descrição matemática do problema de otimização da mochila. Nesse caso, em

uma árvore do espaço de buscas, cada nó representa uma solução parcial que é associada a um valor limite (upper-bound) para a função objetivo em sua projeção a uma solução completa [Ziviani 2010]. Apenas os nós que possuírem um upper-bound melhor que a melhor solução encontrada até então serão expandidos. O branch ou poda produz uma melhoria no tempo de execução a depender da instância e da heurística usada para a construção da árvore[Levitin 2012].

$$\text{Maximize} \quad \sum_{j=1}^{n} v_j x_j$$

$$\text{Sujeito a} \quad \sum_{j=1}^{n} w_j x_j \leq W$$

$$0 \leq x_j \leq 1 \quad \text{for } j = 1, \ldots, n.$$

Figure 1. Representação matemática do Problema da Mochila.
Fonte: [Levitin 2012]

Este trabalho propõe uma solução para o problema da mochila com conflitos (PMC). Nesse caso, as restrições são pares de itens conflitantes os quais não podem coexistir na solução.

A seção seguinte apresenta descreve a metodologia utilizada, seguida da apresentação dos resultados.

2. Metodologia

Dado um conjunto de n itens, a árvore de busca por soluções se baseou na exploração dos itens em ordem de densidade de lucro ($= lucro/peso$). A Figura 2 abaixo ilustra um exemplo para n = 4.

O subconjunto referente à solução parcial em cada nó deve ser registrada. Os itens inclusos podem ser registrados em uma bitstring. Para o exemplo da Figura 2, a solução ótima gerada (completa) é representada como 1010, pois: inclui o item 1, não inclui o 2, inclui o 3 e não inclui o item 4. Esse dois itens totalizam lucro (V) de 65 e peso igual a 9 o que obedece à capacidade máxima da mochila de 10. Observe que esta numeração dos itens na árvore é referente à ordenação por densidade de lucro (vide tabela abaixo na Figura 2). A descoberta de uma solução (completa) viável pode atualizar o valor da melhor solução encontrada até o momento. É com base no valor da melhor solução atual que se faz o branch (ou poda). Nesse caso, se o limite superior (ub) do nó for inferior ao valor da melhor solução encontrada a busca por nós filhos é encerrada. Esta poda foi realizada após a descoberta da solução ótima, do nó 8, para os nós 6 e 2. É este o fator que torna a busca mais eficiente. No entanto, não há garantias em termos do custo do algoritmo uma vez que não se sabe quantos dos 2^n nós folha (ou soluções completas) serão avaliados.

Outro aspecto a considerar é que o PMC inclui restrições para itens conflitantes. A inserção de um novo item à mochila não pode conflitar com os já existentes na solução parcial. Dadas as tuplas de itens conflitantes $(item_i, item_j)$ informadas pelo problema,

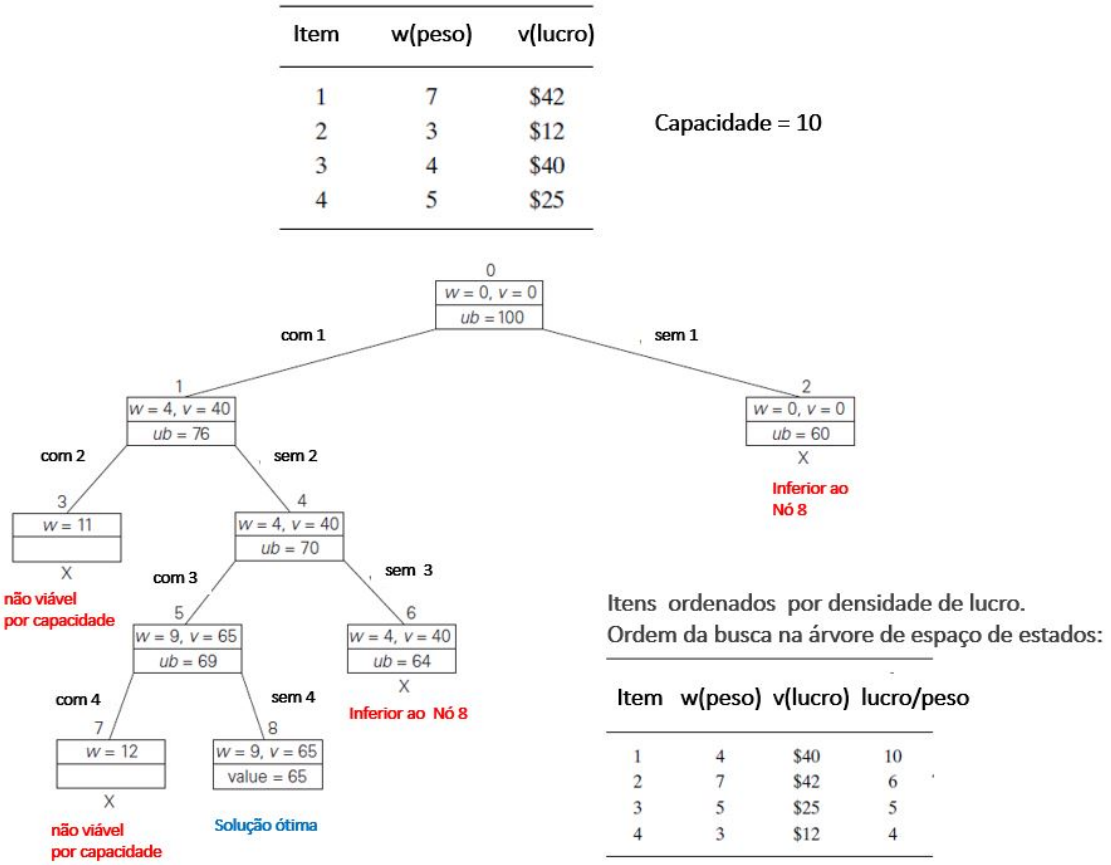

Figure 2. Árvore de espaço de busca em ordem de densidade de lucro (=v/w). Fonte: Adaptado de [Levitin 2012]

uma matriz é preenchida para acesso rápido à restrição. Cada tupla $(item_i, item_j)$ gera um valor *true* para as duas células: linha i, coluna j e linha j, coluna i.

O algoritmo proposto realiza uma busca em profundidade, a partir da raiz. Em cada nó, a busca prossegue primeiro para o nó da esquerda que inclui o próximo item de maior retorno (densidade de lucro). Em seguida a busca segue para o nó à direita que não inclui o próximo item mais lucrativo.

A Figura 3 apresenta um pseudocódigo da busca em cada nó, não raiz.

3. Experimentos

Os testes foram realizados em máquina Intel Core i7, processador 3612QM de 2.1GHz e 4 núcleos de processamento. Esse com cache de 256 KB para L1, L2 de 1 MB e L3 de 6MB, 8 GB de mémória. O programa foi elaborado em Java, compilado e executado no Windows 10.

Foram utilizadas 20 instâncias com tamanhos de entrada (número de itens) variando de 50 á 1000, enviadas por email (Prof. Haroldo). O relatório de saída produzido pela execução é exibido na Figura 4. Os tempos de execução ficaram abaixo de 5 segundos, longe de ultrapassar o limite da máquina de 689 segundos.

O código do programa pode ser executado a partir do $.jar$ enviado, como abaixo

```
procedure: knapsackSolver(iLevel, flagWithItem, nodeSolution, nodeWeight, nodeValue)

    //Verifica se último item em nodeSolution conflita com anteriores
    if (flagWithItem && !hasItensConflicts(nodeSolution))
        return


    //Termina busca em caso da capacidade excedida
    if (nodeWeight > C)
        return

    //Verifica se é nó folha (solução completa)
    if (reachedLeaf(i_level))
        if (nodeValue > bestSolution){}
            bestSolution = valueNode
            bestSolutionItens = nodeSolution
        }
        return

    //Calcula upperBound do nó
    ub = v + (C - nodeWeight)(v[i+1]/w[i+1])

    //Verifica poda (branch)
    if (ub < bestSolution)
        return

    //Continua busca incluindo próximo item
    knapsackSolver(iLevel+1, true, nodeSolution+"1", nodeWeight+w[i+1], nodeValue+v[i+1] ){

    //Continua busca excluindo o próximo item
    knapsackSolver(iLevel+1, false, nodeSolution+"0", nodeWeight, nodeValue)
```

Figure 3. Pseudocódigo busca em produndidade para PMC
Fonte: Elaborado pela autora

indicado:

java -jar knapsack-1.0.jar $< arqInst_1 > .. < arqInst_n >< arqSolucao >$

A Figura 5 resume os tempos de execução e valores para a melhor solução encontrada em cada instância de teste.

4. Considerações finais

Os experimentos realizados demonstraram boa performance do código. As soluções produzidas obedecem a capacidade da mochila e os conflitos entre itens informados em cada instância.

É importante observar que a heurística empregada prioriza a permanência dos itens de maior densidade de lucro na mochila em situações de conflito entre itens. Isso ocorre uma vez que a busca é feita em ordem decrescente de densidade de lucro. Um novo item apenas é inserido à solução se não produzir conflito com os anteriores de maior razão lucro/peso.

References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press.

Levitin, A. (2012). *Introduction to the design analysis of algorithms*. Pearson, New Jersey, 3 ed. edition.

Ziviani, N. (2010). *Projeto de Algoritmos com implementações em Java e C++*. Cengage Learning Edições Ltda.

```
n=  50;  Lucro total:  18,00; Itens:                                9 10 24; Total de itens:  3; Tempo:  5,99090 ms
n= 100;  Lucro total:  33,00; Itens:                             12 24 36 61; Total de itens:  4; Tempo:  3,09210 ms
n= 150;  Lucro total:  55,00; Itens:                   19 30 46 77 144 149; Total de itens:  6; Tempo:  9,14530 ms
n= 200;  Lucro total:  23,00; Itens:                      12 35 52 177 194; Total de itens:  5; Tempo:  7,26160 ms
n= 250;  Lucro total:  37,00; Itens:                      38 55 85 180 185; Total de itens:  5; Tempo: 36,06980 ms
n= 300;  Lucro total:  35,00; Itens:          33 111 119 196 221 253 270 297; Total de itens:  8; Tempo: 10,52770 ms
n= 350;  Lucro total:  39,00; Itens:                   6 43 59 243 312 332; Total de itens:  6; Tempo: 46,73010 ms
n= 400;  Lucro total:  20,00; Itens:                         64 70 172 262; Total de itens:  4; Tempo: 19,58110 ms
n= 450;  Lucro total:  34,00; Itens:                    69 88 104 177 350; Total de itens:  5; Tempo: 23,98370 ms
n= 500;  Lucro total:  26,00; Itens:                25 199 224 303 431 499; Total de itens:  6; Tempo: 36,67020 ms
n= 550;  Lucro total:  22,00; Itens:                      248 349 527 547; Total de itens:  4; Tempo: 8565,59350 ms
n= 600;  Lucro total:  29,00; Itens:                   296 433 482 504 550; Total de itens:  5; Tempo: 62,32610 ms
n= 650;  Lucro total:  23,00; Itens:                   185 242 247 511 614; Total de itens:  5; Tempo: 33,47540 ms
n= 700;  Lucro total:  47,00; Itens:             196 395 442 455 503 515; Total de itens:  6; Tempo: 204,98720 ms
n= 750;  Lucro total:  49,00; Itens:       123 227 315 378 502 670 682 699; Total de itens:  8; Tempo: 113,70030 ms
n= 800;  Lucro total:  30,00; Itens:          162 364 372 652 726 731 759; Total de itens:  7; Tempo: 41,30640 ms
n= 850;  Lucro total:  66,00; Itens:    76 201 202 379 448 541 704 740 813; Total de itens:  9; Tempo: 659,61230 ms
n= 900;  Lucro total:  42,00; Itens:          123 380 398 433 455 746 770; Total de itens:  7; Tempo: 551,17200 ms
n= 950;  Lucro total:  35,00; Itens:        3 162 726 738 749 807 814 923; Total de itens:  8; Tempo: 85,92850 ms
n= 1000; Lucro total:  14,00; Itens:                      448 877 982; Total de itens:  3; Tempo: 4430,22880 ms
```

Figure 4. Arquivo de saída gerado pelo programa: 20 instâncias do PMC
Fonte: Elaborado pela autora

Número de itens (N)	Tempo de execução em milissegundos	Lucro total
50	5,9909	18
100	3,0921	33
150	9,1453	55
200	7,2616	23
250	36,0698	37
300	10,5277	35
350	46,7301	39
400	19,5811	20
450	23,9837	34
500	36,6702	26
550	8565,5935	22
600	62,3261	29
650	33,4754	23
700	204,9872	47
750	113,7003	49
800	41,3064	30
850	659,6123	66
900	551,172	42
950	85,9285	35
1000	4430,2288	14

Figure 5. Tamanho das instâncias, melhor solução obtidas e tempos de execução
Fonte: Elaborado pela autora