UECE - Mestrado Acadêmico em Ciência da Computação
Trabalho da Disciplina: Projeto e Análise de Algoritmos

"Aplicação de Algoritmo Guloso por Maior Valor para a Otimização do Problema da Mochila"

Antonia Raiane Santos Araujo Cruz
raianeti@gmail.com

Antonio Alex de Souza
aasouzaconsult@gmail.com

Wanessa de Caldas Teotônio
wanessacaldast@gmail.com

25 de abril de 2018

Resumo

Neste trabalho, será abordado o problema de otimização combinatória conhecido como Problema da Mochila. O problema é integrante da classe NP-difícil e pode ser enunciado considerando uma situação onde é necessário carregar uma mochila contendo um conjunto de objetos de pesos e valores diferentes. Entretanto, essa mochila possui capacidade limitada. Desse modo, a solução para o problema consiste em maximizar o valor total contido na mochila, de forma que a soma dos pesos dos objetos não ultrapassem o limite suportado. O objetivo deste trabalho é apresentar a implementação de uma heurística gulosa pelo maior valor para otimizar o problema da mochila. Com base nas instâncias estudadas, os resultados obtidos foram abaixo do ótimo conhecido.

Introdução

O Problema da Mochila (Knapsack problem), consiste em um cenário com um conjunto de itens (objetos) a serem colocados em uma mochila. Esse problema faz parte de uma classe de problemas que frequentemente são abordados em estudos de otimização combinatória, além de ser um dos principais problemas da área de Pesquisa Operacional, visto que existem muitas situações, onde se busca maximizar o lucro, que podem ser reduzidas ao resolver este problema [YHSA05].

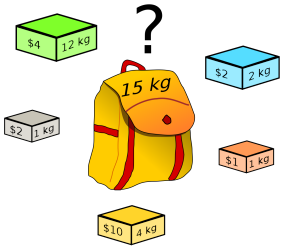

Figura 1: Ilustração do problema da mochila. [Ama13]

A origem do nome, problema da mochila, se deu da necessidade de carregar uma mochila de capacidade limitada, com uma coleção de itens de pesos e valores distintos e que tem como objetivo ocupar a mochila com o maior valor possível, não ultrapassando a sua capacidade máxima. Assim, toda mochila possui uma restrição quanto à sua capacidade, onde a soma do peso dos itens selecionados deve respeitar essa capacidade e ao mesmo tempo maximizar o seu valor total (Imagem 1). [YHSA05].

1 Descrição do Problema

Em síntese, o problema da mochila consiste em escolher um conjunto de itens, onde cada item possui um peso e um valor agregado, de forma que os itens de maior valor associado sejam preferencialmente escolhidos, porém sem ultrapassar a capacidade máxima da mochila, ou seja, dado um conjunto X_n de n itens, representados por $X_n = \{1, 2, ..., n\}$, cada item i de X_n tem um peso w_i e valor v_i ($w_i > 0$ e $v_i > 0$). Dado que sua otimização é determinar um subconjunto Y de X_n tal que a soma dos pesos dos elementos de Y seja a maior possível, mas que estes itens possam ser alocados em um mochila com capacidade W predefinida, como mostra a Equação 1.

$$max \sum_{i=1}^{n} v_i x_i, \text{ sujeito a } \sum_{i=1}^{n} w_i x_i \leq W \tag{1}$$

O Problema da Mochila é integrante da classe NP-difícil, que são problemas complexos de natureza combinatória para os quais não se conhecem algoritmos polinomiais capazes de obter a solução exata, o que exige grande esforço computacional de algoritmos exatos. Esta complexidade tem motivado a busca por heurísticas com o intuito de alcançar melhores resultados em um menor tempo computacional [CV13].

Uma das estratégias para resolver esse problema é a utilização de algoritmos gulosos. Esses algoritmos escolhem a opção que parece ser a melhor no momento (escolha ótima) e esperam que dessa forma consiga chegar a uma solução ótima global [RD04]. Entretanto, os algoritmos gulosos nem sempre apresentam uma solução ótima, pois as decisões são tomadas apenas considerando as informações disponíveis no momento, sem se preocupar com efeitos futuros. Desse modo, não é possível reconsiderar alternativas melhores e desfazer decisões tomadas. Com base nisso, o objetivo deste trabalho é apresentar a implementação de uma heurística gulosa pelo maior valor para otimizar o problema da mochila.

Este artigo está estruturado da seguinte forma: esta introdução, onde descrevemos o Problema da Mochila, suas variantes e alguns trabalhos relacionados; a seção 2 fornece uma breve descrição sobre Heurísticas, Algoritmo Guloso, o Pseudo-código do algoritmo proposto, além da complexidade do algoritmo para o Problema da Mochila e para a Heurística Gulosa proposta; na seção 3 relacionamos os experimentos e resultados obtidos; e por fim, a seção 4, que discorre as considerações finais e sugestões de trabalhos futuros provenientes desta pesquisa.

1.1 Variantes do Problema

Segundo [YHSA05], o Problema da Mochila é amplamente estudado, possuindo diversas variantes agrupadas que podem ser chamadas de famílias, ou classes de mochilas. "Ele caracteriza-se basicamente pela escolha de um subconjunto de itens que irá otimizar um objetivo. Para tanto, cada item deve possuir um "peso"e uma "utilidade". Por peso entende-se um custo, uma penalidade a ser paga, pelo uso do item, por exemplo, quando se escolhe arquivos para compor uma mídia o peso representa o tamanho do arquivo. A utilidade é o lucro, um benefício, que a escolha do item contribuirá no objetivo do problema. Toda mochila possui uma restrição quanto a

sua capacidade, de fato, a soma do peso dos itens selecionados deve respeitar a capacidade da mochila."

[Mor 2007] enuncia que existem outros tipos de Problema da Mochila, com muitas variações nas restrições e condições de execução (Problema da Mochila Compartimentada, Problema da Mochila Multidimensional, que irão ser comentadas a seguir). É interessante analisar, entretanto, três variações do Problema da Mochila que podem ser obtidas adicionando, cada uma, uma pequena restrição no Problema da Mochila original, são elas: O Problema da Mochila Limitado, o Problema da Mochila Ilimitado e o Problema da Mochila 0-1.

- **Problema da Mochila Limitado**: um assaltante está furtando uma residência. Na residência, o número de itens que pode ser levado do mesmo tipo tende a ser limitado; é improvável, por exemplo, que existam mais do que três ou quatro televisores. Assim, cada item pode ser incluído no somatório um número limitado de vezes.

- **Problema da Mochila Ilimitado**: neste caso não existem restrições quanto ao número máximo que se pode carregar de um respectivo item (no caso do assalto, poderiam levar 5 ou 10 televisores, caso tivesse na casa).

- **Problema da Mochila 0/1**: no mesmo cenário dos anteriores, o ladrão, desta vez, atenta em roubar uma loja de raros vasos chineses. Tal loja tem apenas um tipo de cada vaso e, deste modo, a decisão do ladrão é entre levar o item ou não.

1.2 Trabalhos relacionados

Os problemas de Otimização consistem em encontrar a melhor combinação dentre um conjunto de variáveis dado para maximizar ou minimizar uma função, geralmente chamada de função objetivo ou função custo [Bec08], e representa a área da Matemática Aplicada que se concentra em calcular valores ótimos ou próximos de ótimo, para variáveis de decisão de acordo com algum critério de avaliação, ao mesmo tempo que satisfazem restrições de um modelo matemático [Nak04]. Preocupa-se com o resultado ótimo para determinados problemas que envolvam a maximização ou minimização em função de critérios já definidos.

Em [BR17] é proposto três soluções para Knapsack problem, com algoritmos distintos, para um problema relativo à possibilidade de encontrar um conjunto de itens nos quais a soma de seus pesos não ultrapasse um valor dado e que a soma da suas utilidades seja a maior possível. Para a primeira solução implementou-se um algoritmo de força bruta que compara todas as possibilidades de preenchimento da mochila sem ultrapassar o limite máximo. O algoritmo devolve um resultado correto, porém para um conjunto de valores muito grande, o tempo gasto é exponencial. Em um segundo algoritmo, utilizou-se programação dinâmica que, se comparado com o primeiro, determina o resultado correto em menos tempo, entretanto, também quando aumentado a capacidade da mochila, o tempo fica bem próximo de ser exponencial. E como um terceiro algoritmo utilizou-se uma heurística de aproximação baseada em programação gulosa, que pode ou não determinar a solução, porém leva em consideração uma taxa de aproximação com um tempo na ordem de $O(n)$ ou $O(n^2)$.

O Problema da Mochila pode ser otimizada por diversas abordagens. [SR14] propuseram às abordagens Força Bruta, Programação Dinâmica, Algoritmo Guloso e Branch and Bound para solucionar o problema de otimização combinatória, aplicando-as ao Problema da Mochila 0/1. Para este trabalho, avaliou-se às soluções propostas apenas pela abordagem por Algoritmo Guloso. Para o problema dado, considerou-se três estratégias de seleção gulosa, a primeira equivale em selecionar os itens de maior valor com o objetivo de aumentar o valor da mochila rapidamente; a segunda foram selecionados os itens de menor peso buscando levar a maior quantidade de itens possíveis; e por fim, na terceira optaram pelos itens com maior razão valor/peso, com a expectativa de levar os itens mais valiosos e leves.

Assim como implementado por [BR17], a proposta deste artigo é implementar uma heurística gulosa que devolva um valor próximo de ótimo para a abordagem Problema da Mochila 0/1 otimizando pelo maior valor, de forma que se comparado com os trabalho supracitados, é possível classificar às abordagens, em relação a este trabalho como similares. Comparando-se ao primeiro trabalho, em sua terceira implementação percebe-se que a complexidade do algoritmo é da ordem de $O(n)$ ou $O(n^2)$ assim como a heurística proposta por esta pesquisa, que tem complexidade $O(n)$.

2 O Algoritmo Guloso

Para [Ben12], ao pensar em um problema altamente combinatório, uma opção seria analisar todas as combinações possíveis em busca da melhor. Se o problema possui um universo de dados pequeno, esta é a maneira correta de buscar a melhor solução. Mas, os problemas reais, normalmente, possuem um número de combinações muito grande, o que torna inviável a análise de todas as combinações, uma vez que o tempo computacional torna-se muito elevado, para esses casos, entram as heurísticas.

Segundo [Ben12] as heurísticas ou algoritmos heurísticos são um conjunto de regras e métodos que coordenam à descoberta, à invenção e à resolução do problema com nível de complexidade mais elevado, em tempo computacional razoável. Para [Bue09], o objetivo dos métodos heurísticos nem sempre é a solução ótima de um problema, de forma que, inicialmente uma solução viável, baseia-se em consecutivas aproximações direcionadas a um ponto ótimo. Estes métodos visam encontrar as melhores soluções possíveis para o problema, e não soluções exatas, perfeitas, definitivas.

Neste trabalho utilizou-se uma heurística por meio de algoritmos gulosos por maior valor, que consiste em resolver o problema passo a passo, ordenando os itens e, a cada passo, faz-se a melhor escolha no momento, sem se preocupar com o impacto futuro desta escolha e sem desfazer escolhas passadas, até um resultado mais próximo de ótimo.

Geralmente, esse tipo de estratégia tem a desvantagem de ser incapaz de escapar de ótimos locais, pois não permite desfazer passos da busca. Entretanto, uma das vantagens é que algoritmos que a implementam normalmente confluem relativamente mais rápido, o que torna esta abordagem interessante quando se desejam soluções viáveis rápidas, além de serem relativamente simples de entender e implementar. [Mor15]

2.1 Complexidade do Algoritmos

A complexidade do Problema da Mochila é exponencial, representado por $O(2^n)$, isso quer dizer que a medida que n aumenta, o fator que estiver sendo estudado (tempo ou espaço) cresce exponencialmente. Problemas dessa natureza são considerados intratáveis (com tempos computacionais não aceitáveis e/ou não conhecidos). Por conta disso, são utilizadas heurísticas, que buscam uma solução próxima ao ótimo em um tempo computacional aceitável.

Na implementação do algoritmo proposto, baseado em algoritmos gulosos, utilizou-se o método *sorted()* com o objetivo de obter um vetor ordenado pela ordem decrescente do valor e crescente do peso de cada item, ou seja, o item de maior valor e menor peso ocupa o início da posição do vetor, o segundo item de maior valor e menor peso ocupa a segunda posição e, assim sucessivamente. A complexidade do método *sorted()* é da ordem de $O(n\ log\ n)$. Após realizar a ordenação, o algoritmo irá pegar o máximo de elementos ordenados, iniciando pelo primeiro, até que a capacidade da mochila seja alcançada. Se o peso de um item da mochila ultrapassa a capacidade máxima, este item é descartado e o algoritmo testa o próximo item de maior valor do vetor, até que sejam colocados o máximo de itens dentro da mochila. Sem considerar o tempo de ordenação da entrada, o algoritmo proposto executa a estratégia gulosa em tempo $O(n)$.

Desse modo, considerando que o custo de ordenação é $O(n\ log\ n)$ e que o algoritmo guloso executa n vezes, concluímos, através da soma das notações, que o tempo total de execução é $O(n)$. Portanto, a ordem de complexidade do algoritmo proposto é de $O(n)$.

2.2 Pseudo-Código do Algoritmo

Essa seção apresenta o pseudo-código do algoritmo guloso por maior valor proposto para o Problema da Mochila.

Algorithm 1: ProblemaMochilaGulosoMaiorValor

Input: Capacidade da Mochila $\langle W \rangle$
Input: Valores $\langle v_1, v_2, \ldots, v_n \rangle$
Input: Pesos $\langle w_1, w_2, \ldots, w_n \rangle$
$capacidadeTotal \leftarrow 0$
$value \leftarrow 0$
$numItems \leftarrow tamanho(Valores)$

valorProbMochGuloso ← *Ordenação Decrescente dos Valores[1] (Maior Valor)*
** comentário: valorProbMochGuloso[i][0] - Refere-se a Pesos
** comentário: valorProbMochGuloso[i][1] - Refere-se a Valores

while $Capacidade > 0$ ***and*** $numItems > 0$ **do**
    idx ← 0
    if *valorProbMochGuloso[idx][1]* ≤ *Capacidade* **then**
        value += valorProbMochGuloso[idx][0]
        capacidadeTotal += valorProbMochGuloso[idx][1]
        capacidade -= valorProbMochGuloso[idx][1]
    end

    Output: *Para cada item (objeto)*
    - Item alocado na Mochila
    - Valor alocado do item
    - Valor acumulado na Mochila
    - Capacidade alocada do item
    - Capacidade acumulada na Mochila
    ——————————————
    - Vetor de Valores da Mochila
    - Vetor de Pesos da Mochila
    - Itens Alocados

    return *value*
end

O código do algoritmo proposto se encontra em no seguinte no GitHub [1]

[1] *Link: https://github.com/aasouzaconsult/DiversosMachineLearning/blob/master/KnapsackProblem.py*

3 Experimentos Computacionais

O algoritmo foi submetido à análise tanto com a utilização de pequenas como de grandes instâncias. Para as pequenas, considerou-se 5, 10, 20, 30 e 50 quantidade de itens, definidas aleatoriamente pelos testadores. Para as grandes instâncias, de 1.000, 2.500, 5.000, 7.500 e 10.000, utilizou-se valores obtidos do site *OR-LIBRARY* [2]

3.1 Formato do Arquivo de Entrada

As entradas para o algoritmo são: a capacidade da mochila e os pesos e valores de cada item. A capacidade da mochila é armazenada na variável "capacidade". Os valores dos itens são armazenados no vetor "valores" e os pesos no vetor "pesos".

Desse modo, é um exemplo de entrada válida para o algoritmo:

```
capacidade = 85
valores = [60, 100, 120, 80, 30]
pesos = [20, 50, 30, 10, 40]
```

Como saída, para a entrada de exemplo, o algoritmo fornece os seguintes valores:

```
--------------------------------------------------------------------------------
                          Número de Itens - 5
                      Capacidade da Mochila:  - 85
 Valor - Decrescente:  - [[120, 30], [100, 50], [80, 10], [60, 20], [30, 40]]
--------------------------------------------------------------------------------
                       Item alocado na Mochila:  1
                       Valor alocado do item:  120
                    Valor acumulado na Mochila:  120.0
                      Capacidade alocada do item:  30
                  Capacidade acumulada na Mochila:  30.0
--------------------------------------------------------------------------------
                    Num itens alocados na Mochila:  2
                       Valor alocado do item:  100
                    Valor acumulado na Mochila:  220.0
                      Capacidade alocada do item:  50
                  Capacidade acumulada na Mochila:  80.0
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
                Vetor de Valores da Mochila:  [120, 100]
                  Vetor de Pesos da Mochila:  [30, 50]
                           Itens Alocados:  2
--------------------------------------------------------------------------------
                          Executado em 0.001s.
--------------------------------------------------------------------------------
```

3.2 Resultados Obtidos

Os resultados obtidos com o uso do Algoritmo Guloso para Maior Valor proposto, não foram satisfatórios, ficando muito abaixo dos valores "ótimos"das instâncias da *OR-LIBRARY*, como podemos ver na Tabela 1 abaixo.

[2] *Link OR-LIBRARY: http://people.brunel.ac.uk/ mastjjb/jeb/info.html*

Tabela 1: Resultados obtidos por instância

Instâncias Comparativas				Solução	Solução Obtida			Aproximação
Nº	Referência	n	C	"Ótima"	Itens Alocados	tempo(s)	Inicial/Melhor	Gap
1	KP1000	1000	500	12800	10	0.006	1500	8,53
2	KP2500	2500	295	8560	17	0.072	1697	5,04
3	KP5000	5000	2500	37660	120	0.152	11885	3,16
4	KP7500	7500	3804	58165	184	0.173	18214	3,19
5	KP10000	10000	5000	77621	237	0.427	23504	3,30
					Média das aproximações com resultados conhecidos (%) ⇒			4,64
Nº	Outras	n	C	Itens Alocados	Mínima	Média	Inicial/Melhor	tempo(s)
6	TE05	5	85	2	220	220	220	<0.000
7	TE10	10	50	7	42	42	42	<0.000
8	TE20	20	50	6	504	504	504	0.001
9	TE30	30	120	12	1066	1066	1066	0.010
10	TE50	50	180	20	1697	1697	1697	0.031

Na parte superior da Tabela 1, encontra-se os resultados comparativos do algoritmo proposto com as instâncias da *OR-LIBRARY* e na parte inferior, alguns testes com instâncias menores.Os campos em comum são: n (número de itens), C (capacidade da mochila), Itens Alocados (Quantidade de n alocados) e tempo (segundos).

Na parte superior, temos a Solução "Ótima"que é disponibilizada pela própria *OR-LIBRARY* para cada uma de suas instâncias, assim como as colunas: Melhor e GAP. A coluna: Melhor, retorna o resultado do nosso algoritmo proposto, que por ser guloso e ordenado por maior valor, sempre retorna o mesmo resultado (independente de quantas vezes for executado), já a coluna: GAP, foi calculada como base na relação entre as colunas: Melhor e "Ótima", onde tivemos um GAP médio de 4,64.

Na parte inferior, além dos campos em comum, temos as colunas Mínima, Média e Melhor. E pelo mesmo motivo exposto acima (de ser um algoritmo guloso e ordenado por maior valor) os resultados dessas colunas são todos iguais.

A Figura 2 abaixo, mostra um comparativo entre o Tamanho da Instância (número de itens) pelo tempo de execução do Algoritmo Guloso Prosposto para aquela Instância, vemos que "naturalmente"o tempo cresce conforme a quantidade de itens da instância aumenta, mas observem que o ponto 4 (1000 itens em 0.006s) teve um tempo menor de que em instâncias menores (30 itens em 0.010s). Isso ocorreu porque na instância de 30 itens (TE30), o algoritmo teve que percorrer mais vezes o vetor em busca de itens que o levasse à capacidade total da mochila.

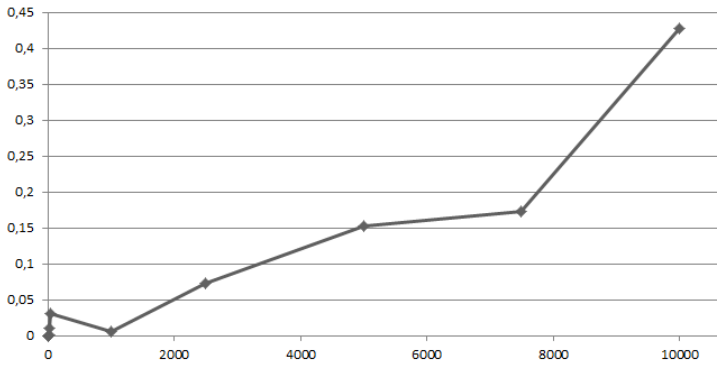

Figura 2: Tamanho da Instância (eixo x) X Tempo de Execução (eixo y)

4 Considerações Finais

Este trabalho apresentou uma discussão sobre o Problema da Mochila, sua complexidade e a implementação de um algoritmo guloso para otimizar a sua solução, considerando maximizar o valor. Os problemas de Otimização buscam soluções que exigem a determinação de valores máximos ou mínimos absolutos das funções que os representam. O problema da Mochila pode ser considerado um problema de Otimização, pois as soluções encontradas são as melhores possíveis. Para tanto, foi fundamental contextualizar e caracterizar o problema proposto, além de apresentar os principais trabalhos relacionais e os resultados obtidos.

Para a solução do Problema da Mochila aqui tratado, implementou-se uma heurística gulosa pelo maior valor para otimizar o problema da mochila. Os experimentos foram realizados em instâncias de tamanhos pequenos, nos quais o resultado chegam próximo ao ótimo, porém, para experimentos com instâncias maiores, obteve-se um resultado bem abaixo do ótimo esperado.

Ciente de que este artigo alcançou o objetivo proposto, mas que a heurística proposta não obteve resultados próximos ao ótimo, propõe-se como trabalhos futuros, a implementação de outras heurísticas para solucionar o mesmo problema e um comparação entre as estas abordagens, a fim de obter uma tabela comparativa para análise dos algoritmos que apresentem resultados mais próximos ao ótimo. Além de implementar um algoritmo guloso para otimizar a solução com Mochilas Múltiplas/Multidimensional.

Referências

[Ama13] Silvio Roberto Martins Amarante. "utilizando o problema de m´˘ltiplas mochilas para modelar o problema de alocação de m´quinas virtuais em computação nas nuvens". Master's thesis, Universidade Estadual do Cear´, 2013.

[Bec08] José Carlos Becceneri. Meta-heurísticas e otimização combinatória: Aplicações em problemas ambientais. *INPE, Sao José dos Campos*, 2008.

[Ben12] Paula Francis Benevides. Aplicação de heurísticas e metaheurísticas para o problema do caixeiro viajante em um problema real de roterização de veículos. 2012.

[BR17] Felipe R. S. Barbosa and Leonardo dos Reis Vilela Rafael. Problema da mochila, Setembro 2017.

[Bue09] Fabrício Bueno. Métodos heurísticos. *Teoria e implementações. Araranguá: IFSC*, 2009.

[CV13] Rebeca Marcilio Araújo Capdeville and Dalessandro Soares Vianna. Heurísticas grasp para o problema de alocação de pontos de acesso em uma rede sem fio em ambiente indoor. *Sistemas & Gestão*, 8(1):86–93, 2013.

[Mor15] João Carlos Heringer Moreira. Uma abordagem com multi-mochilas multidimensionais para o problema de alocação de ações de redução de perdas na distribuição de energia. Master's thesis, Universidade Federal do Espírito Santo, 2015.

[Nak04] Paulo Hiroaqui Ruiz Nakashima. Otimização de processos de produção de petróleo via injeção contínua de gás. *Proposta de Tese, Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina*, 2004.

[RD04] Anderson Rocha and Leyza Baldo Dorini. Algoritmos gulosos: definic oes e aplicac oes. *Campinas, SP*, page 42, 2004.

[SR14] Éfren Lopes de Souza and Erik Alexander Landim Rafael. Abordagens para resolver o problema da mochila 0/1. *REVISTA IGAPÓ-Revista de Educação Ciência e Tecnologia do IFAM*, 3, 2014.

[YHSA05] Horacio Yanasse, Robinson Hoto, Fernando Spolador, and Marcos Arenales. Um modelo linear para o problema da mochila compartimentada. *IX ON PCE–Oficina Nacional de problemas de Corte e Empacotamento*, 2005.