Projeto 03 - Binpacking e algoritmos aproximados

Arthur Wallace Silva Lopes

[1]Instituto Federal de Brasília (IFB)

1. Introdução

O problema *Binpacking* consiste em, dado um conjunto de itens de diferentes pesos p e um cesto(*bin*) de capacidade c, definir o menor número possível de cestos necessários para armazenar todos esses itens. Entretanto, a versão ótima da solução para esse problema é **NP-Completa**, ou seja, não existe, até o presente momento, uma solução polinomial capaz de resolvê-lo completamente com 100% de assertividade. Dessa forma, é possível atingir soluções com assertividades satisfatórias, em relação à resolução ótima, em tempos de execuções razoáveis por meio de algoritmos aproximados, ou heurísticos.

Este projeto visa avaliar a implementação de alguns algoritmos aproximados mais conhecidos para se aproximar das resoluções do problema *Binpacking*, são eles: *First-fit*, *Next-fit* e *Best-fit* e comparar suas respectivas assertividades em relação à resolução ótima e também os tempos e custos de execução para os piores casos. Isso será realizado por meio da utilização dos arquivos de dados fornecidos contendo a quantidade total de itens, a capacidade de cada cesto e o peso de cada um dos itens a serem inseridos.

Dessa forma, ao fim desse trabalho, busca-se:

- Demonstrar que esses algoritmos possuem tempos viáveis de execução para o pior caso, por meio da análise assintótica;
- Provar que os algoritmos citados possuem um raio de aproximação 2 em relação às soluções ótimas;
- Comparar as assertividades desses algoritmos com a solução ótima verificando se o raio de aproximação se confirma e, também, demonstrar que os algoritmos aproximados podem ser uma possível alternativa para resolver, de forma satisfatória, o problema *Binpacking*.

2. Abordagens aproximadas para o Problema BIN-PACKING-OPTM

2.1. First-Fit

A primeira abordagem utilizada será o método **First-Fit**. Neste método, a lista com os itens não terá sua ordem alterada e cada um dos itens de peso p será alocado no primeiro cesto (o de menor índice) capaz de acomodá-lo, ou seja, onde a diferença entre a capacidade restante e o peso do item seja maior ou igual a zero ($C - p >= 0$) e, caso ainda não exista esse cesto, um novo será criado e o item será alocado nele.

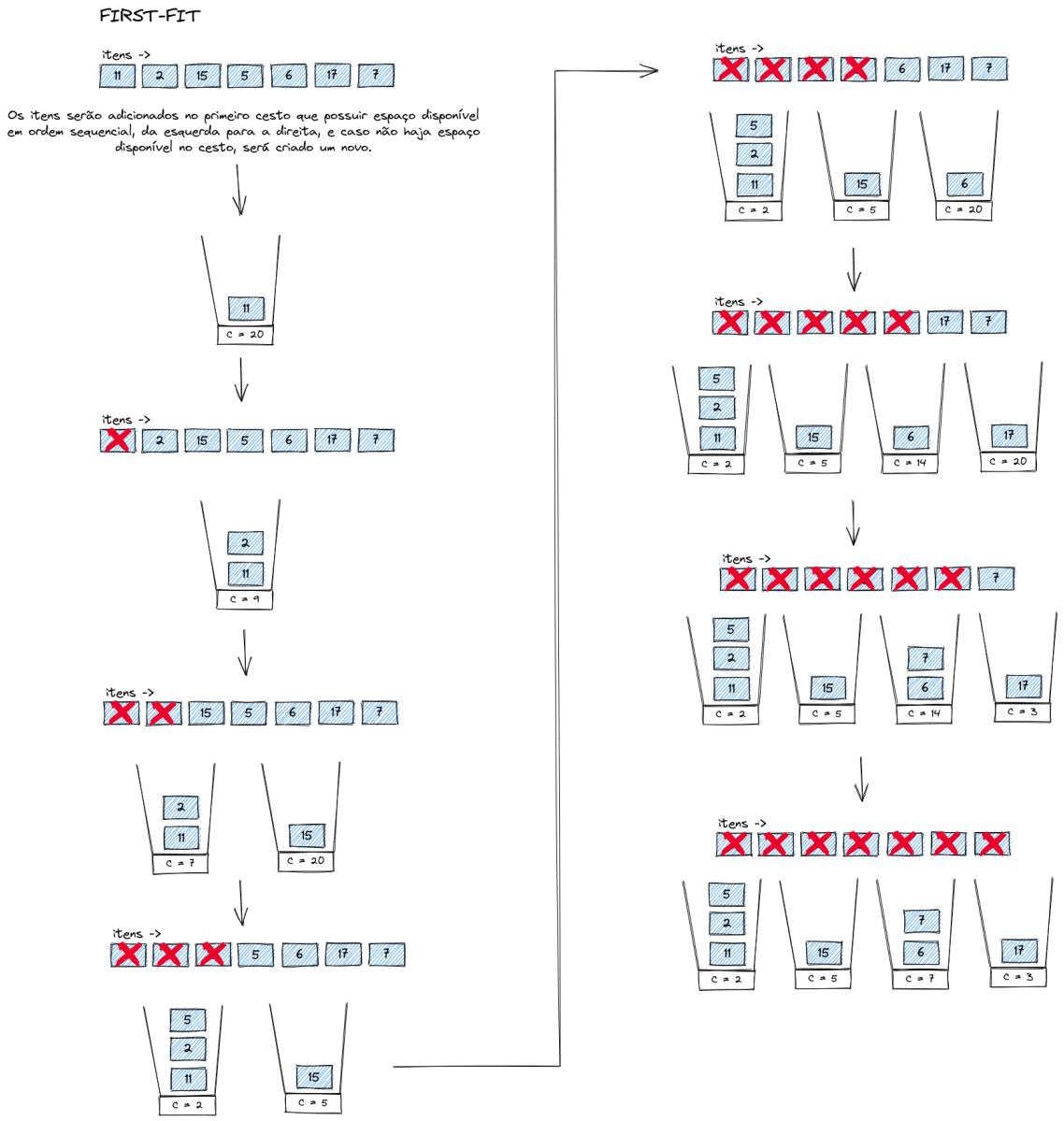

Figura 1. First-Fit exemplo

Ao analisar a complexidade de execução do algoritmo por meio da abordagem gulosa priorizando o **número de itens a serem inseridos**, conseguimos, através da análise assintótica, determinar o custo total desse método ao fim dessas iterações.

Primeiramente, é necessário realizar a leitura dos arquivos de conjunto de dados que possuirá custo $\Theta(n)$ para ser executada. Logo após, os itens serão ordenados utilizando o método *sorted* do Python, que possuirá custo $\Theta(n \log n)$ para ser executado.

Por fim, será realizada uma iteração sobre esses dados ordenados a fim de comparar o peso de cada item com a capacidade da mochila e, caso ainda exista espaço disponível, adicioná-lo na mochila. Essa iteração possuirá custo $\Theta(n)$ para ser executada.

Com isso, temos que o custo total dessa execução será de:

$$\begin{aligned} \Theta(n) + \Theta(n \log n) + \Theta(n) &= \Theta(n + n \log n + n) \\ \Theta(2n + n \log n) &= \Theta(n \log n) \\ \Theta(n \log n) & \end{aligned} \tag{1}$$

2.2. Next-Fit

Nessa abordagem, o algoritmo vai, a cada iteração, buscar o item de melhor benefício/custo. Para alcançar esse objetivo, ordena-se os itens de forma decrescente quanto ao seus valores de benefício/custo e, caso a inserção do item não exceda a capacidade da mochila, retira o item da lista e o adiciona na mochila.

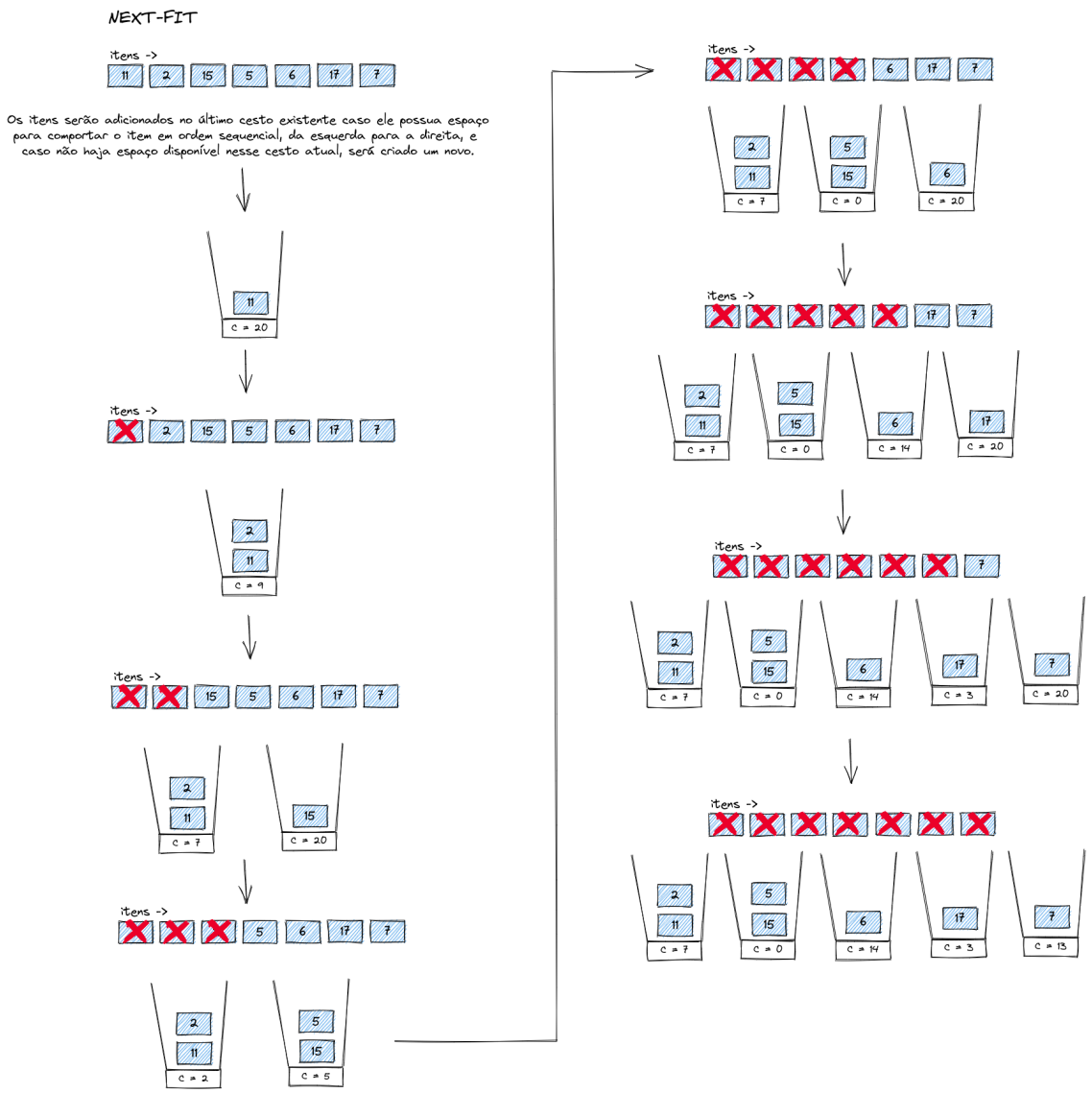

Figura 2. Next-Fit exemplo

Ao analisar a complexidade de execução do algoritmo por meio da abordagem gulosa priorizando os itens de **melhor relação de benefício/custo**, conseguimos, através da análise assintótica, determinar o custo total desse método ao fim dessas iterações.

Primeiramente, é necessário realizar a leitura dos arquivos de conjunto de dados que possuirá custo $\mathbf{\Theta(n)}$ para ser executada. Logo após, os custos benefícios ($valor/peso$) dos itens de cada categoria serão calculados e ordenados de forma decrescente utilizando o método *sorted* do Python, que possuirá custo $\mathbf{\Theta(n \log n)}$ para ser executado.

Por fim, será realizada uma iteração sobre esses dados ordenados a fim de comparar o peso de cada item com a capacidade da mochila e, caso ainda exista espaço disponível, adicioná-lo na mochila. Essa iteração possuirá custo $\mathbf{\Theta(n)}$ para ser executada.

Com isso, temos que o custo total dessa execução será de:

$$\begin{aligned} \Theta(n) + \Theta(n \log n) + \Theta(n) &= \Theta(n + n \log n + n) \\ \Theta(2n + n \log n) &= \Theta(n \log n) \\ &\Theta(n \log n) \end{aligned} \tag{2}$$

2.3. Best-Fit

A segunda abordagem será usando programação dinâmica, que se baseia na premissa de evitar a repetição de resoluções para o mesmo problema múltiplas vezes, usando matrizes para armazenar os valores ótimos obtidos a cada iteração e utilizá-los para consulta nas iterações subsequentes.

Se analisarmos como seria uma estrutura ótima do problema atual, veremos que a solução do problema é composta por soluções de sub instâncias do mesmo problema. A partir dessas informações, conseguimos definir a estrutura de uma solução ótima para o problema através da seguinte relação de recorrência:

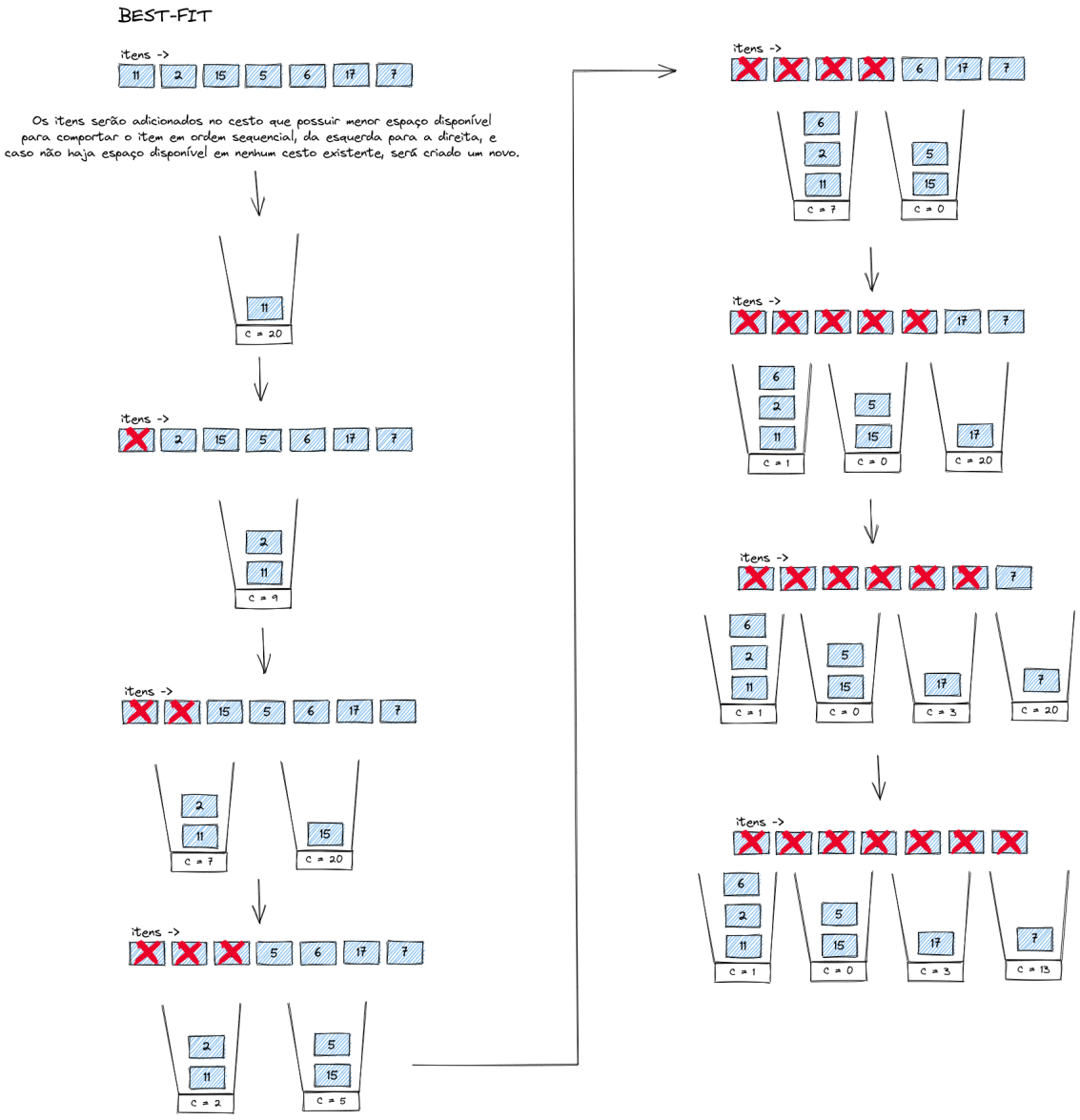

Figura 3. Best-Fit exemplo

Essa recursividade se dará da seguinte forma:

- Inicialmente, teremos o caso base em que para todas as capacidades a mochila terá peso igual a 0. As iterações seguirão a premissa booleana de que, ou um item é adicionado por inteiro na mochila ou ele é descartado;
- Logo após, caso o peso do item na iteração seja maior do que a capacidade restante da mochila, ele não será adicionado e se repetirá o melhor caso anterior;
- Caso o item caiba na mochila, é realizada a comparação entre os valores possíveis nas duas situações e o **maior** desses valores será utilizado:
    1. não adicionar o item à mochila
    2. colocá-lo na mochila, adicionando o seu valor à soma do melhor resultado com o peso restante
- Essa recursividade será executada até que todo o vetor de itens seja percorrido.

Dessa forma, ao fim da construção da matriz teremos os valores ótimos possíveis a cada iteração e no último elemento da matriz será armazenado o maior valor possível para essa mochila. Com essa matriz construída, também é possível determinar quais itens devem ser selecionados para alcançar essa solução ótima, da seguinte forma:

- Iniciamos a observação pelo último elemento da matriz;
- Caso o elemento seja igual ao valor imediatamente superior a ele, o item **não será adicionado** na mochila e a validação seguirá para a linha anterior;
- Caso o item seja diferente do valor imediatamente superior, o item **será adicionado** na mochila e a validação seguirá para a linha anterior na coluna referente ao peso restante da mochila, ou seja, o peso anterior menos o peso do novo item adicionado;
- Essa busca se repetirá até que se alcance o primeiro elemento dessa matriz.

Ao analisar a complexidade de execução do algoritmo por meio da **Programação Dinâmica**, conseguimos, por meio da análise assintótica, determinar o custo total desse método ao fim dessas iterações.

Primeiramente, é necessário realizar iterações sobre o vetor de itens, que possuirão custo $\mathbf{\Theta(n)}$ para serem executadas, sendo *n* a quantidade de itens. Logo após, será realizado um novo loop, partindo do 0 até atingir a capacidade máxima da mochila (*C*), que possuirá custo $\mathbf{\Theta(C)}$ para ser executado.

Dessa forma, como teremos dois loops aninhados, as complexidades de cada um deles será multiplicada. Assim teremos que:

$$\begin{aligned} \Theta(n) * \Theta(C) &= \Theta(n * C) \\ &\quad \Theta(n * C) \end{aligned} \tag{3}$$

3. Materiais e Métodos

Para a realização desse projeto, utilizamos a linguagem *Python* para implementação dos algoritmos e fizemos uso dos três categorias de datasets fornecidos como referência:

- Categoria 1: possui **fraca** correlação entre peso e valor de cada item
- Categoria 2: possui **média** correlação entre o peso e valor de cada item
- Categoria 3: possui **forte** correlação entre o peso e o valor de cada item

Cada categoria possui arquivos com as respectivas quantidades de itens a serem avaliados (100, 200, 500, 1000, 2000, 5000, 10000). Cada um desses arquivos possuem: o tamanho da mochila, os valores e pesos de cada item e a resolução ótima o problema. Essas categorias, serão utilizadas para demonstrar o quanto a correlação entre os pesos e valores dos itens afetam diretamente a eficiência e assertividade do algoritmo guloso.

Para realizar os testes, estamos executando o algoritmos por 3 vezes e, a partir dessas execuções, podemos calcular a média dos tempos de execução e também a proximidade alcançada pelos métodos em relação à resolução ótima e gerar as tabelas e gráficos de comparações.

4. Análise de Dados

4.1. Abordagem gulosa

4.1.1. Priorizando o número de itens a serem inseridos

Ao analisar a assertividade alcançada nas diferentes categorias utilizando a Abordagem Gulosa Priorizando o número de itens, podemos perceber, com base nos dados de coleta apresentados na tabela abaixo, que:

- Na **Categoria 1**, os resultados obtidos chegam a uma média de aproximação de **90%** dos valores ótimos. Isso pode se dar pelo fato de que os itens dessa categoria possuírem uma **fraca correlação** entre valor e peso e, assim, os itens são bem dispersos, e a chance de se utilizar itens de baixo peso com um alto valor é relativamente alta;
- Na **Categoria 2**, os resultados obtidos chegam a uma média de aproximação de **64.5%** dos valores ótimos. Essa diferença é percebida pois, como a correlação entre valor e peso desses itens é **média**, os itens são um pouco menos dispersos em relação à categoria 1, dificultado a seleção de itens de baixo peso com um valor agregado alto;
- Na **Categoria 3**, os resultados obtidos chegam a uma média de aproximação de **99.5%** dos valores ótimos, se mostrando a melhor das categorias. Isso ocorre pois nessa categoria, a correlação entre valor e peso é **forte**, sendo possível alcançar valores bem próximos ao resultado ótimo.

Em relação ao tempo de execução, foi possível observar que a categoria não possuíra relevância significante para alterar os resultados e sim o tamanho das entradas a serem analisadas.

4.1.2. Priorizando os itens de melhor relação de benefício/custo

Ao analisar a assertividade alcançada utilizando a Abordagem Gulosa Priorizando a relação de beneficio/custo, podemos perceber, com base nos dados de coleta apresentados na tabela abaixo, que as diferentes correlações entre as categorias não possui grande impacto nos resultados obtidos, por o fator a ser considerado será a relação de benefício/custo entre os itens, apresentando, assim, resultados bem próximos aos resultados ótimos em todas as categorias.

Em relação ao tempo de execução, foi possível observar que a categoria não possuíra relevância significante para alterar os resultados e sim o tamanho das entradas a serem analisadas.

4.2. Programação dinâmica

Na Programação Dinâmica, a assertividade da solução sempre atingirá os resultados ótimos pois irá fazer, a cada iteração, a análise baseada nos melhores resultados das iterações anteriores. Entretanto, os tempos de execução para essa solução serão bem mais

altos, atingindo, nos casos testados, até **250 segundos** para resolver a melhor solução para uma mochila de tamanho 10000.

O código para esse projeto está disponível em: `https://github.com/MatheusNSantiago/problema-da-mochila-booleana/blob/master/main.ipynb`

5. Considerações finais

Por fim, após a realização desse projeto, podemos chegar às seguintes conclusões:

- Com base nos gráficos apresentados e na análise assintótica, foi possível perceber que a abordagem gulosa atinge resultados satisfatórios se tratando de assertividade em relação aos resultados ótimos e, além disso, possui um tempo de execução bem baixo em relação à abordagem pela Programação Dinâmica. Logo, ao se buscar um custo/benefício maior entre a precisão de acerto e o tempo de execução, a Abordagem Gulosa se prova o excelente opção a ser utilizada.
- Também pudemos perceber o efeito que as diferentes correlações entre as categorias causam ao analisar a Abordagem Gulosa priorizando a quantidade de itens, a qual interfere diretamente nas precisões obtidas. Isso indica que a escolha do parâmetro a ser otimizado é fundamental para um resultado satisfatório.

Dessa forma, ao confrontar a análise teórica dos métodos com os resultados obtidos, mostra-se evidente que as complexidades de execução de ambos os métodos são bem diferentes, já que, enquanto a abordagem guloso será dependente apenas da quantidade de itens (n), possuindo complexidade $\mathbf{\Theta}(\mathbf{n}\log\mathbf{n})$, a abordagem dinâmica sempre terá influência tanto da quantidade de itens (n) quanto da capacidade da mochila (C), possuindo complexidade de $\mathbf{\Theta}(\mathbf{n}\cdot\mathbf{C})$.

6. Referências Bibliográficas

THE KNAPSACK Problem. [S. l.], 23 nov. 2011. Disponível em: https://courses.csail.mit.edu/6.006/fall11/rec/rec21_knapsack.pdf. Acesso em: 7 jul. 2022.

MOCHILA booleana. [S. l.], 21 nov. 2020. Disponível em: https://www.ime.usp.br/ pf/analise_de_algoritmos/aulas/mochila-bool.html. Acesso em: 7 jul. 2022.