Algoritmos para o Problema da Mochila Booleana: Experimento e Avaliação

Giovanni Lucas O. da Silva[1]

[1]Instituto Federal de Brasília Campus Taguatinga (IFB)
72146-050 – Brasília – DF – Brazil

giovanni61540@estudante.ifb.edu.br

Abstract. *This technical report presents a comparative study between different algorithmic approaches for solving the Boolean knapsack problem, including two greedy algorithms and an algorithm based on dynamic programming. The greedy algorithms were designed with different approaches: the first prioritizes the maximization of the number of selected items, while the second uses the cost-benefit ratio to determine the inclusion of items. In addition, a dynamic programming algorithm was implemented as an exact solution to the problem. The experimental analysis highlights the advantages and disadvantages of each approach, considering both the quality of the obtained solutions and the computational efficiency. The results show that dynamic programming provides exact solutions, while the greedy algorithms present competitive performance.*

Resumo. *Este relatório técnico apresenta um estudo comparativo entre diferentes abordagens algorítmicas para a resolução do problema da mochila booleana, incluindo dois algoritmos gulosos e um algoritmo baseado em programação dinâmica. Os algoritmos gulosos foram elaborados com enfoques distintos: o primeiro prioriza a maximização do número de itens selecionados, enquanto o segundo utiliza a razão custo-benefício para determinar a inclusão dos itens. Além disso, um algoritmo de programação dinâmica foi implementado como uma solução exata para o problema. A análise experimental/algorítmica destaca as vantagens e desvantagens de cada abordagem, considerando tanto a qualidade das soluções obtidas quanto a eficiência computacional. Os resultados mostram que a programação dinâmica oferece soluções exatas, enquanto os algoritmos gulosos apresentam desempenho competitivo.*

1. Introdução

O problema da mochila booleana é um dos problemas clássicos em otimização combinatória e teoria da computação, amplamente estudado devido à sua relevância prática e teórica [Martello e Toth 1990]. Esse problema pode ser descrito da seguinte maneira: dado um conjunto de itens, cada um com um peso e um valor associado, e uma mochila com capacidade limitada, o objetivo é selecionar uma combinação de itens que maximize o valor total sem exceder a capacidade máxima da mochila. No caso da versão booleana, cada item pode ser incluído ou excluído, ou seja, a escolha de um item é binária (sim ou não).

Este problema surge em diversas áreas, incluindo logística, finanças, gerenciamento de recursos, e até mesmo em aplicações como a seleção de projetos[Martello e

Toth 1990]. Em muitos casos, ele modela situações onde é necessário fazer escolhas sob restrições, buscando um equilíbrio entre custo e benefício.

Resolver o problema da mochila é computacionalmente desafiador, sendo um exemplo típico de problema NP-completo, o que significa que não se conhece, até o momento, uma solução eficiente para todos os casos (veja [Garey e Johnson 1979] para maiores explicações). Diversos algoritmos foram propostos para lidar com o problema da mochila booleana, variando desde métodos exatos, como a programação dinâmica, até abordagens heurísticas, como algoritmos gulosos. Enquanto os algoritmos exatos garantem a solução ótima, eles são muitas vezes inviáveis para instâncias grandes devido ao custo computacional. Em contrapartida, os algoritmos heurísticos, como os gulosos, oferecem soluções aproximadas de forma mais eficiente como veremos.

Neste relatório, será realizada uma análise experimental de três algoritmos para a solução do problema da mochila booleana: dois algoritmos gulosos, um focado na maximização do número de itens e outro na razão custo-benefício, e um algoritmo de programação dinâmica. A experimentação foi conduzida em instâncias com três níveis de correlação entre itens da mochila e valor(fraca, média e forte), com o intuito de avaliar o impacto dessa característica no desempenho dos algoritmos.

2. Abordagens para o Problema da Mochila Booleana

Neste experimento, foram implementadas três abordagens para a solução do problema da mochila booleana: dois algoritmos gulosos que são mais simples de implementar e entender e um algoritmo exato baseado em programação dinâmica, que utiliza técnicas mais aprimoradas para achar a solução ótima. A seguir, cada abordagem será descrita e ilustrada com exemplos.

2.1. Algoritmo Guloso: Maximização do Número de Itens

O primeiro algoritmo guloso visa maximizar a quantidade de itens na mochila, priorizando a inclusão dos itens mais leves. A estratégia consiste em ordenar os itens pelo peso em ordem crescente e, em seguida, adicioná-los à mochila até atingir a capacidade máxima.

Exemplo: Suponha que temos uma mochila com capacidade de 10 unidades de peso e os seguintes itens:

Tabela 1. Itens para o problema da mochila booleana

Item	Peso	Valor
A	3	40
B	4	30
C	5	50
D	2	20

O algoritmo ordena os itens pelo peso, resultando na ordem: D (2), A (3), B (4), C (5). Em seguida, os itens são adicionados à mochila:

- Adiciona o item D (peso 2, capacidade restante 8)
- Adiciona o item A (peso 3, capacidade restante 5)

- Adiciona o item B (peso 4, capacidade restante 1)

A solução final contém os itens D, A e B, com um valor total de 90, o último iten não é adicionado por que a capacidade da mochila não permite.

2.2. Algoritmo Guloso: Maximização da Razão Custo-Benefício

O segundo algoritmo guloso prioriza a razão *custo-benefício*, isto é, o valor dividido pelo peso dos itens. O algoritmo ordena os itens por essa razão e os adiciona à mochila conforme a capacidade disponível.

Exemplo: Usando o mesmo conjunto de itens, as razões valor/peso dos itens são:

Tabela 2. Razão valor/peso dos itens

Item	Peso	Valor	Razão (Valor/Peso)
A	3	40	13.33
B	4	30	7.5
C	5	50	10.0
D	2	20	10.0

O algoritmo ordena os itens pela razão valor/peso: A (13.33), C (10.0), D (10.0), B (7.5). Em seguida, os itens são adicionados à mochila:

- Adiciona o item A (peso 3, capacidade restante 7)
- Adiciona o item C (peso 5, capacidade restante 2)
- Adiciona o item D (peso 2, capacidade restante 0)

A solução contém os itens A, C e D, com um valor total de 110, uma solução superior ao algoritmo anterior.

2.3. Programação Dinâmica

A abordagem de programação dinâmica resolve o problema de maneira exata. A solução é baseada na construção de uma tabela $V(i, w)$[Hristakeva and Shrestha 2005], onde i é o número de itens considerados e w é a capacidade da mochila. A relação de recorrência utilizada é a seguinte [Souza e Rafael, 2009]:

$$V(i,w) = \begin{cases} V(i-1,w) & \text{se } peso[i] > w \\ \max(V(i-1,w), V(i-1,w-peso[i]) + valor[i]) & \text{se } peso[i] \leq w \end{cases}$$

Você precisa inicializar a tabela de valores V com base em condições base. Por exemplo:

$$V(0,w) = 0 \text{ para qualquer capacidade } w,$$

porque sem itens, o valor máximo é 0.

Preencha a tabela V para todos os itens e capacidades possíveis usando a relação de recorrência. O valor máximo que pode ser obtido com todos os itens e a capacidade total da mochila é dado por:

$$V(n,W)$$

onde n é o número total de itens e W é a capacidade total da mochila.

Essa abordagem percorre todas as combinações possíveis de itens e capacidades de mochila de maneira eficiente, usando a tabela para não precisar verificar todas as combinações explicitamente, o que é essencial para resolver problemas de otimização como o problema da mochila.

Exemplo: Com os mesmos itens e uma capacidade de 10, a tabela de programação dinâmica é preenchida da seguinte forma:

Tabela 3. Tabela da programação dinâmica

Itens/Capacidade	0	1	2	3	4	5	6	7	8	9	10
Sem item	0	0	0	0	0	0	0	0	0	0	0
Item D (2, 20)	0	0	20	20	20	20	20	20	20	20	20
Item A (3, 40)	0	0	20	40	40	60	60	60	60	60	60
Item B (4, 30)	0	0	20	40	40	60	60	70	70	90	90
Item C (5, 50)	0	0	20	40	40	60	60	70	70	90	110

A tabela mostra que, com uma capacidade de 10, o valor máximo possível é 110, obtido ao incluir os itens A, C e D. Esta abordagem garante a solução ótima, porém, com um custo computacional maior pois teremos de resolver todos os caminhos possíveis para achar a melhor solução.

3. Materiais e Metodologia

Esta seção descreve os materiais e a metodologia utilizados para a execução dos experimentos sobre o problema da mochila booleana.

3.1. Materiais

Os algoritmos foram implementados na linguagem C, e o código-fonte está disponível no seguinte repositório GitHub:

`https://github.com/Giovanni-LOS/problema_mochila`

Para garantir o desempenho, foi utilizado o compilador GCC, com a seguinte opção de compilação $-O3$ para otimização de execução do código.

Além disso, foi desenvolvido um script automatizado para realizar a compilação, execução e coleta dos resultados, permitindo maior facilidade e reprodutibilidade dos experimentos.

3.2. Metodologia

Os experimentos foram conduzidos para comparar diferentes abordagens ao problema da mochila booleana: dois algoritmos gulosos e um algoritmo baseado em programação dinâmica.

Para a ordenação dos itens, utilizou-se o algoritmo *Merge Sort*, conhecido por sua eficiência em ordenação estável com complexidade $\Theta(n \log n)$ [Paulo Feofiloff 2019]. A ordenação foi aplicada de acordo com a estratégia de cada algoritmo:

- Para o algoritmo guloso de maximização do número de itens, os itens foram ordenados de acordo com o peso crescente.
- Para o algoritmo guloso de maximização da razão custo-benefício, os itens foram ordenados pela razão valor/peso, de forma decrescente.

A metodologia de teste envolveu a execução dos algoritmos com três níveis de correlação entre os itens: fraca, média e forte. Cada nível de correlação define a relação entre o valor e o peso dos itens, sendo testado com diferentes conjuntos de dados presentes no proprio repositório.

3.3. Execução dos Experimentos

A execução dos experimentos foi realizada em ambiente Linux com um processador Intel i7 de 11° geração de 4.7GHz, utilizando o terminal para rodar os scripts de compilação e execução dos algoritmos. Para garantir a reprodutibilidade um script principal foi criado para automatizar a execução, compilação e coleta de dados, otimizando o processo de experimentação e assegurando que os parâmetros de entrada fossem consistentes entre os diferentes experimentos. O script está configurado para rodar os programas cinco vezes e tirar a média aritmética de cada teste do diretório $large\_scale$. onde $knapPI\_1$ se refere as correlações fracas, $knapPI\_2$ se refere as correlações médias e $knapPI\_3$ se refere as correlações fortes.

Para a reprodução do experimento é necessário que seja feito o clone do repositório em um ambiente Linux e que entre no diretório pelo terminal e insira o comando $./script.sh$ para que seja executado o script. Um arquivo chamado *resultados.txt* será criado com todos os resultados dos experimentos.

4. Análise de Dados e Algoritmos

Nesta seção, serão apresentados os dados obtidos durante os experimentos com os algoritmos gulosos e o algoritmo de programação dinâmica. As tabelas e gráficos ilustram o desempenho de cada abordagem para os diferentes níveis de correlação entre o valor e o peso dos itens. Além disso, será realizada a análise assintótica dos algoritmos utilizados, justificando os resultados observados.

4.1. Apresentação e discussão dos Dados

Os experimentos foram conduzidos em três cenários distintos de correlação: fraca, média e forte. As tabelas e gráficos apresentam os valores obtidos para o desempenho dos algoritmos, em termos comparação com a solução ótima e tempo de execução. Para a correlação fraca de valor e peso temos:

Tabela 4. Métricas de qualidade das soluções dos algoritmos gulosos e programação dinâmica para diferentes quantidades de itens em correlação fraca.

Número de Itens	Métrica Guloso Máximo de Itens	Métrica Guloso Custo Benefício	Métrica Programação Dinâmica
100	0,936154	0,963923	1,000000
200	0,901851	0,999021	1
500	0,872821	0,999203	1
1000	0,871897	0,997853	1
2000	0,884059	0,999295	1
5000	0,887458	0,999718	1
10000	0,879992	0,999925	1

Tabela 5. Tempos de execução dos algoritmos gulosos e programação dinâmica para diferentes quantidades de itens em correlação fraca.

Número de Itens	Tempo Guloso Máximo de Itens (s)	Tempo Guloso Custo Benefício (s)	Tempo Programação Dinâmica (s)
100	0,000006	0,000014	0,001082
200	0,00003	0,00003	0,002998
500	0,000071	0,000074	0,013247
1000	0,000124	0,000157	0,048031
2000	0,000239	0,000311	0,195431
5000	0,000532	0,000886	1,173653
10000	0,001111	0,001749	4,918722

Para os gráficos temos a linha/coluna azul para o guloso máximo, vermelha para o guloso custo- benefício e a linha/coluna amarela para a programação dinamica.

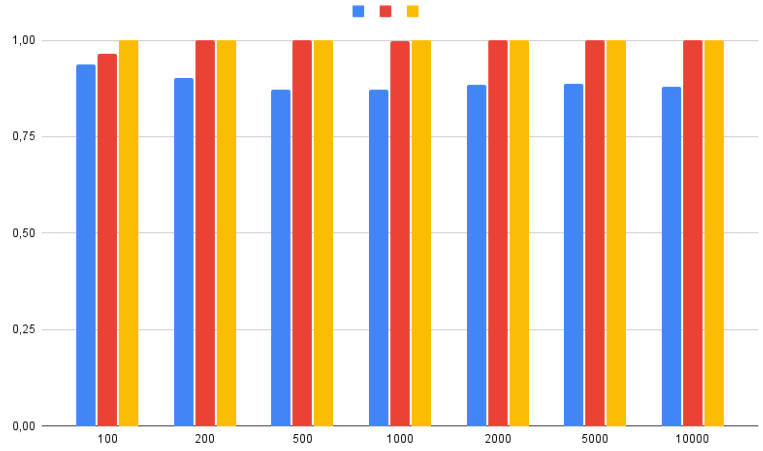

Figura 1. Desempenho qualitativo dos algoritmos para correlação fraca.

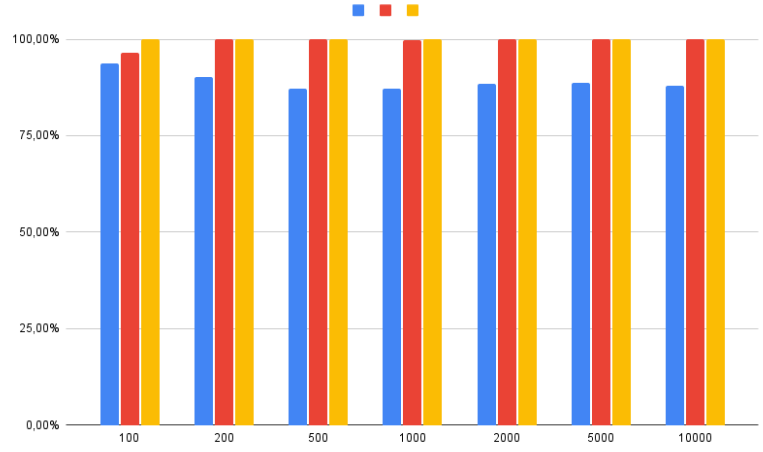

Figura 2. Desempenho qualitativo dos algoritmos para correlação fraca com a escala vertical em porcentagem para melhor visualização.

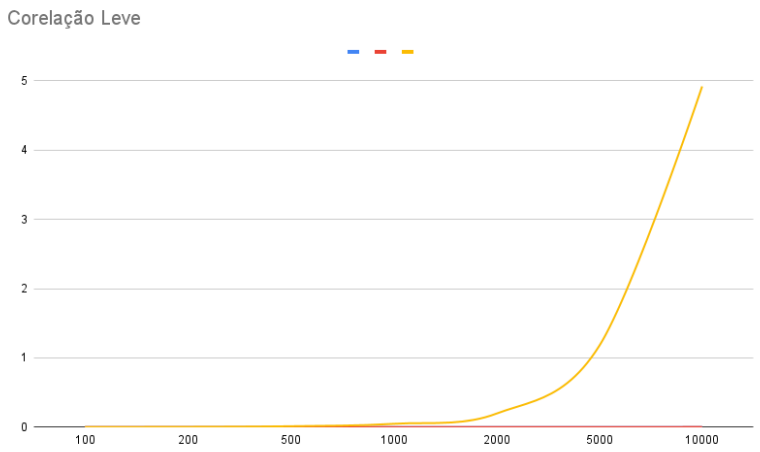

Figura 3. Desempenho temporal em segundos dos algoritmos para correlação fraca.

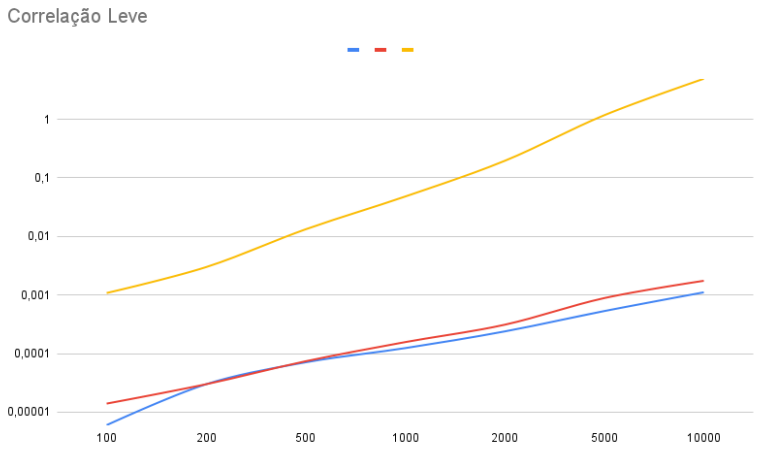

Figura 4. Desempenho temporal em segundos dos algoritmos para correlação fraca com a escala vertical em logarítmo para melhor visualização.

O algoritmo guloso de máximo de itens apresenta as métricas mais baixas entre os três, com valores que variam entre 0,87 e 0,93. O desempenho cai ligeiramente à medida que o número de itens aumenta, embora a diferença não seja muito grande, já o de custo-benefício se aproxima muito da solução ótima (métrica próxima de 1) à medida que o número de itens aumenta. Para 100 itens, ele já atinge 0,963923, e a partir de 200 itens, a métrica ultrapassa 0,999, o que indica que, em cenários de correlação fraca entre os itens, esse método é eficaz em encontrar soluções próximas do ótimo, e a dinâmica oferece a solução ótima, sempre apresenta um valor de métrica igual a 1, independentemente do número de itens. Isso confirma que esse algoritmo encontra a solução exata para o problema da mochila booleana. Na questão de tempo os algoritmos gulosos eles apresentam um tempo bem similar enquanto o algoritmo dinâmico sempre apresenta um tempo bem maior.

Para a correlação média temos:

Tabela 6. Qualidade das soluções dos algoritmos em correlação média

Número de Itens	Métrica Guloso Máximo de Itens	Métrica Guloso Custo Benefício	Métrica Programação Dinâmica
100	0,628137	0,982166	1,000000
200	0,670747	0,981640	1,000000
500	0,680026	0,996934	1,000000
1000	0,679629	0,999337	1,000000
2000	0,689742	0,999280	1,000000
5000	0,633150	0,999887	1,000000
10000	0,655869	0,999956	1,000000

Tabela 7. Tempos de execução dos algoritmos em correlação média

Número de Itens	Tempo Guloso Máximo de itens	Tempo Guloso Custo Benefício	Tempo Programação Dinâmica
100	0,000008	0,000016	0,001016
200	0,000029	0,000030	0,003018
500	0,000076	0,000074	0,014091
1000	0,000150	0,000157	0,050754
2000	0,000218	0,000327	0,206440
5000	0,000547	0,000893	1,195994
10000	0,001682	0,001770	4,766134

Aqui também temos a linha/coluna azul para o guloso máximo, vermelha para o guloso custo- benefício e a linha/coluna amarela para a programação dinamica.

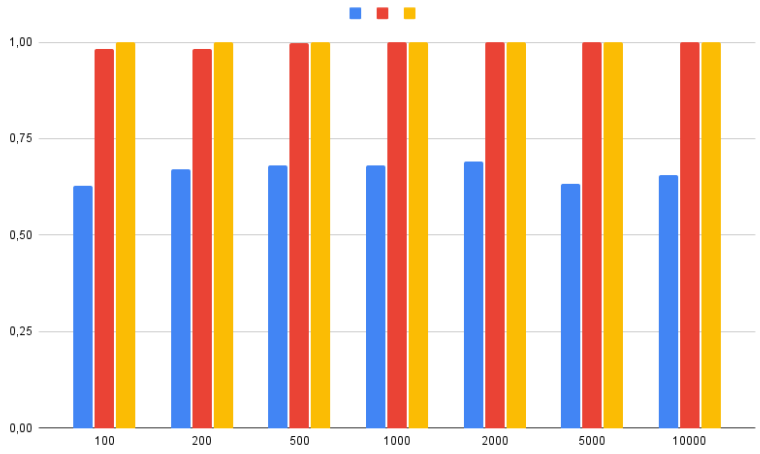

Figura 5. Desempenho qualitativo dos algoritmos para correlação média.

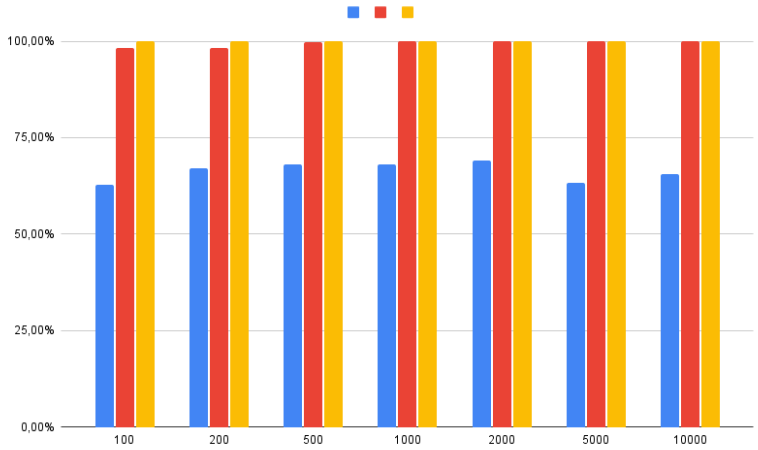

Figura 6. Desempenho qualitativo dos algoritmos para correlação média com a escala vertical em porcentagem para melhor visualização.

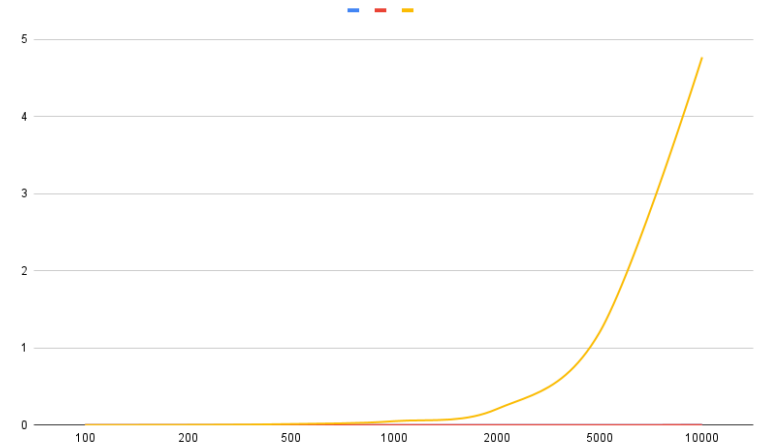

Figura 7. Desempenho temporal em segundos dos algoritmos para correlação média.

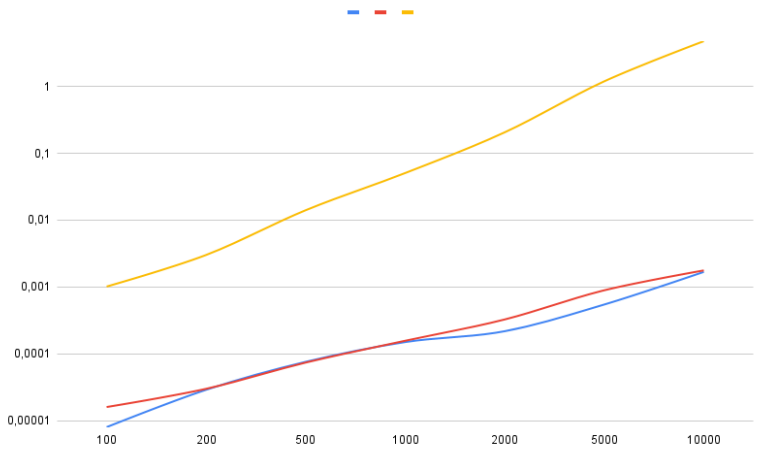

Figura 8. Desempenho temporal em segundos dos algoritmos para correlação média com a escala vertical em logarítmo para melhor visualização.

Para o guloso máximo a métrica de eficácia varia de 0,628 para 100 itens a 0,655 para 10000 itens. A eficácia é inferior à dos outros algoritmos, o que reflete o fato de que o algoritmo de maximização de itens não é indicado em instâncias com correlação média, já o de custo-benefício por sua vez, apresenta uma qualidade de solução muito alta, com métricas acima de 0,98 em todos os casos, chegando a 0,9999 para 10000 itens. Isso indica que ele lida muito bem com a correlação média. A programação dinâmica sempre alcança a solução ótima (métrica de 1,0 em todos os casos), embora seja significativamente mais lenta.

Finalmente para correlações fortes temos:

Tabela 8. Qualidade das soluções dos algoritmos em correlação forte

Número de Itens	Métrica Guloso Máximo de Itens	Métrica Guloso Custo Benefício	Métrica Programação Dinâmica
100	0,990822	0,990822	1,000000
200	0,982202	0,982202	1,000000
500	0,997330	0,997330	1,000000
1000	0,998888	0,998888	1,000000
2000	0,996819	0,996819	1,000000
5000	0,999186	0,999186	1,000000
10000	0,999789	0,999789	1,000000

Tabela 9. Tempos de execução dos algoritmos em correlação forte

Número de Itens	Tempo Guloso Máximo de Itens	Tempo Guloso Custo Benefício	Tempo Programação Dinâmica
100	0,000008	0,000014	0,001198
200	0,000029	0,000028	0,002885
500	0,000069	0,000074	0,012319
1000	0,000144	0,000159	0,050848
2000	0,000218	0,000331	0,201299
5000	0,000596	0,000903	1,15362
10000	0,001263	0,001797	4,815066

Temos a linha/coluna azul para o guloso máximo, vermelha para o guloso custo-benefício e a linha/coluna amarela para a programação dinamica.

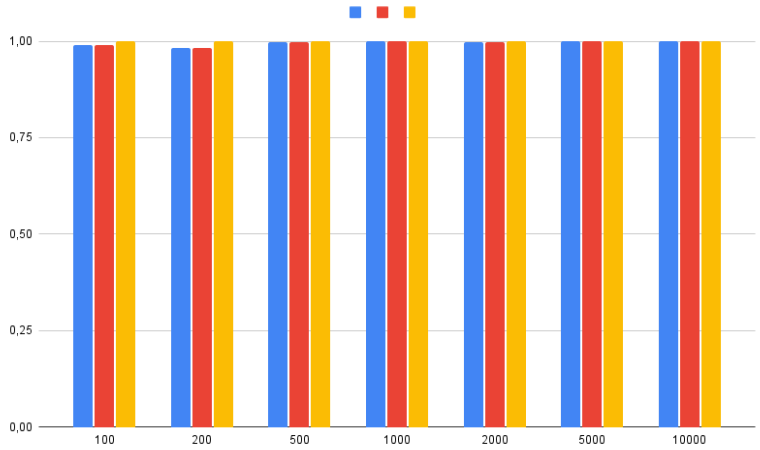

Figura 9. Desempenho qualitativo dos algoritmos para correlação forte.

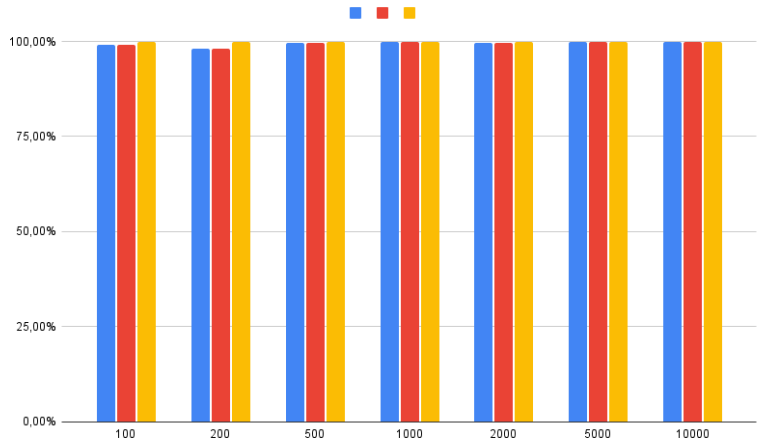

Figura 10. Desempenho qualitativo dos algoritmos para correlação forte com a escala vertical em porcentagem para melhor visualização.

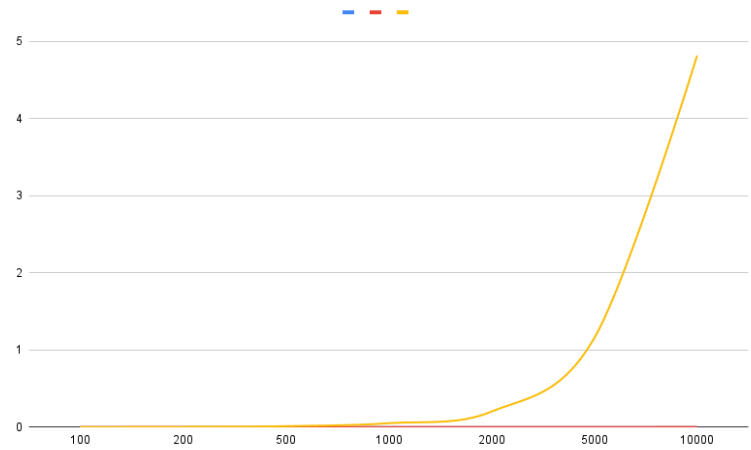

Figura 11. Desempenho temporal em segundos dos algoritmos para correlação forte.

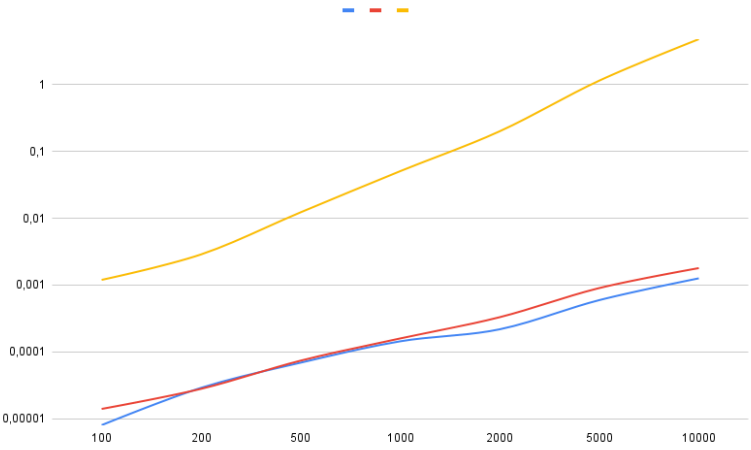

Figura 12. Desempenho temporal em segundos dos algoritmos para correlação forte com a escala vertical em logarítmo para melhor visualização.

Para os gulosos os resultados mostram que a métrica de eficácia do algoritmo é bastante alta, próxima de 1,0 em todos os casos e os dois apresentam identicos resultados variando de 0,990822 para 100 itens a 0,999789 para 10.000 itens e a dinâmica fornece sempre a melhor solução. As linhas de tempo são identica aos outros, onde a dinãmica tende a ser a mais demorada.

4.2. Discussão dos Algoritmos

4.2.1. Desempenho dos Algoritmos Gulosos

Os algoritmos gulosos apresentaram resultados satisfatórios em termos de tempo de execução, com ambos mostrando uma complexidade assintótica de $\Theta(n \log n)$ devido ao uso do algoritmo de ordenação *Merge Sort*. A fase de seleção dos itens, após a

ordenação, é linear, ou seja, $\Theta(n)$. Portanto, a complexidade geral dos algoritmos gulosos é $\Theta(n \log n)$.

No entanto, a qualidade das soluções obtidas varia conforme a estratégia de seleção dos itens. O algoritmo guloso que prioriza o número de itens (maximização de itens) tende a produzir soluções subótimas, principalmente quando a correlação entre valor e peso dos itens é média. Isso ocorre porque o algoritmo ignora o valor dos itens ao priorizar aqueles de menor peso, o que pode levar à inclusão de muitos itens de baixo valor.

Já o algoritmo guloso baseado na razão *valor/peso* apresentou resultados significativamente melhores, especialmente nos cenários de correlação média. Ao priorizar itens de maior valor relativo, ele tende a se aproximar mais da solução ótima em muitos casos. No entanto, como se trata de uma abordagem heurística, não há garantia de que a solução encontrada seja sempre a melhor possível.

4.2.2. Desempenho do Algoritmo de Programação Dinâmica

O algoritmo de programação dinâmica apresentou o melhor desempenho em termos de qualidade da solução, alcançando sempre a solução ótima, independentemente do nível de correlação dos itens. Neste algoritmo a complexidade de tempo é $O(n \cdot W)$, pois todos os arranjos de itens são testados em todas as possibilidades das capacidades[Souza e Rafael, 2009], onde n é o número de itens e W é a capacidade da mochila. Embora essa abordagem seja eficiente para valores moderados de W, seu custo computacional pode se tornar um problema para instâncias com grandes capacidades ou muitos itens. É importante mencionar que precisamos de uma tabela bidimensional para armazenar os valores de todas as chamadas para todos os itens ii e capacidades. O tamanho da tabela é também de complexidade $O(n \cdot W)$.

Como podemos observar nos gráficos, o tempo de execução do algoritmo de programação dinâmica cresce linearmente com o número de itens e a capacidade da mochila, sendo mais lento que os algoritmos gulosos em instâncias maiores. Entretanto, sua capacidade de fornecer a solução ótima faz dele a escolha ideal para situações em que o tempo de execução não é a principal preocupação.

4.2.3. Resumo da Análise Assintótica

Abaixo está um resumo das complexidades assintóticas de cada algoritmo:

- **Algoritmo Guloso (maximização de itens)**: $O(n \log n)$, devido à ordenação dos itens.
- **Algoritmo Guloso (custo-benefício)**: $O(n \log n)$, também dominado pela ordenação dos itens.
- **Programação Dinâmica**: $O(n \cdot W)$, onde W é a capacidade da mochila.

Os resultados obtidos pelos experimentos confirmam a análise teórica de que os algoritmos gulosos são mais rápidos, mas não garantem a solução ótima, enquanto a programação dinâmica, apesar de mais lenta, fornece a melhor solução possível.

4.3. Complexidade ciclomática

O valor da complexidade ciclomática obtida automaticamente pelo site SonarCloud gerou que os algoritmos gulosos apresentaram o mesmo valor ciclomatico de 16 e o de programação dinâmica apresentou o valor 17. Essas metricas mostram que são algoritmos de complexidade de construção parecida, mas não apresenta relação com a complexidade temporal ou acertiva dos algoritmos. "A complexidade ciclomática é definida como a medição da quantidade de lógica de decisão em uma função de código-fonte [NIST235]. Simplificando, quanto mais decisões precisarem ser tomadas no código, mais complexas serão.". [Microsoft 2022]

5. Considerações Finais

Os resultados mostraram que os algoritmos gulosos em termos de tempo de execução são bons, sendo uma excelente escolha para grandes volumes de dados, especialmente quando o tempo de processamento é um fator crítico. No entanto, a qualidade das soluções obtidas por esses algoritmos varia, dependendo da estratégia utilizada e da correlação entre peso e valor dos itens.

A abordagem gulosa com máximo de itens demonstrou a menor qualidade nas soluções, especialmente em cenários com correlação média entre peso e valor, sua estratégia simples de maximizar o número de itens leva a soluções subótimas, com métricas de eficácia inferiores em comparação com outras abordagens como o algoritmo guloso custo-benefício que se destacou como uma solução de muito boa, conseguindo uma qualidade de solução próxima da ótima em quase todos os cenários. Sua eficácia foi boa em casos de correlação baixa e forte, com métricas que atingem até 99,99 da solução ótima, enquanto ainda mantém um tempo de execução muito baixo. Isso faz do algoritmo guloso de custo-benefício uma escolha altamente recomendada em contextos onde é necessário um equilíbrio entre precisão e desempenho.

O algoritmo de programação dinâmica sempre encontrou a solução ótima para o problema da mochila booleana. No entanto, o custo de processamento deste algoritmo aumentou consideravelmente com o tamanho da instância, especialmente para conjuntos de dados maiores, onde o tempo de execução ultrapassou vários segundos. Apesar disso, para problemas de pequeno ou médio porte, onde a solução exata é necessária, a programação dinâmica é a escolha ideal.

Ao contrastar as abordagens, observamos que não existe uma única solução ideal para todos os casos. A escolha do algoritmo depende fortemente dos requisitos do problema, como o tamanho da instância, o grau de correlação entre os itens e a necessidade de encontrar a solução exata. Em geral, para problemas grandes e em cenários onde a solução ótima não é um requisito absoluto, o algoritmo guloso de custo-benefício mostrou-se a alternativa mais equilibrada, oferecendo alta qualidade com excelente desempenho temporal. Por outro lado, a programação dinâmica se destaca como a única abordagem que garante a solução ótima, mas com uma penalidade significativa em tempo de execução para grandes conjuntos de dados.

Portanto, este trabalho demonstra a importância de avaliar o contexto do problema da mochila booleana antes de escolher a abordagem mais adequada, balanceando a necessidade de precisão com as limitações computacionais.

Referências

[1] S. Martello and P. Toth. Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons, Chichester, UK, 1990.

[2] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, San Francisco, CA, 1979.

[3] Éfren Lopes de Souza and Erik Alexander Landim Rafael. *Abordagens para resolver o problema da mochila 0/1*. Universidade Federal do Amazonas (UFAM) and Instituto Federal de Educação, Ciência e Tecnologia do Amazonas (IFAM), 2009.

[4] Paulo Feofiloff. Merge Sort. 2019. Disponível em: `https://www.ime.usp.br/~pf/algoritmos/aulas/mrgsrt.html`. Acesso em: 14 set. 2024.

[5] Hristakeva, M. and Shrestha, D. *Different Approaches to Solve the 0/1 Knapsack Problem*. 2005. Editora Campus, Rio de Janeiro.