Trabalho Prático 2 de Algoritmos 2: Soluções para problemas difíceis

Giovanni Russo Paschoal[1], **João Pedro Wadge**[1], **Wilian Ventura dos Reis**[1]

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) - Belo Horizonte, MG - Brazil

giovanni.russo@dcc.ufmg.br,joaopedrowadge@ufmg.br, wilianv@ufmg.br

Abstract. *The main objective of this work is to implement and compare the results and metrics of the Branch-and-Bound algorithm, which delivers the optimum value, and two other 2-approximative algorithms, which guarantee a value at most twice as bad as the optimum, one using a greedy strategy and the other dynamic programming. As a way of comparing the algorithms implemented, tests were carried out to verify the efficiency of each one and, based on the results found, to understand and define the situation in which each of them should be used. In this sense, it was possible to infer that Branch-and-Bound is more useful in situations where obtaining the optimum is essential, placing less emphasis on time expenditure, while approximations prove to be useful in scenarios where time is an essential factor, with some flexibility as to the quality of the solution. Thus, the main motivation is to understand how the search for the optimal result brings a higher computational cost compared to approximations that compensate for the exact result with a lower cost.*

Resumo. *O principal objetivo deste trabalho é implementar e comparar os resultados e métricas dos algoritmos Branch-and-Bound, que entrega o valor ótimo, e de outros dois algoritmos 2-aproximativos, que garantem um valor, no máximo, duas vezes pior que o ótimo, sendo um utilizando estratégia gulosa e outro programação dinâmica. Como forma de comparar os algoritmos implementados, foram traçados testes para verificar a eficiência de cada um deles, e, a partir do resultado encontrado, entender e definir a situação indicada para o uso de cada um deles. Nesse sentido, por meio destes, foi possível inferir que o Branch-and-Bound apresenta-se mais útil para situações onde obter o ótimo é essencial, colocando-se menor ênfase no gasto de tempo, enquanto que os aproximativos demonstram-se proveitosos em cenários nos quais o tempo é um fator essencial, existindo certa flexibilidade quanto a qualidade da solução. Assim, a principal motivação é entender como a busca do resultado ótimo trás um custo computacional maior se comparado aos aproximativos que compensam o resultado exato com um custo menor.*

1. Introdução

O objetivo deste trabalho prático consiste em analisar o comportamento de diferentes tipos de algoritmo ao solucionar o *Problema da Mochila Binário*. Especificamente, será avaliado o algoritmo *Branch-and-Bound* e soluções 2-aproximativas, uma com a estratégia gulosa e outra com um modelo de programação dinâmica (DP) + *Fully Polynomial-Time*

Approximation Scheme (FPTAS). O problema em análise pertence à classe NP-difícil, isto é, não existe uma solução determinística polinomial conhecida, o que torna a formulação de métodos que fogem de uma abordagem de força bruta essencial. Nesse sentido, a importância de tal estudo se encontra na observação dos resultados práticos desses métodos sobre instâncias variadas do problema, o que visa entender o compromisso que a escolha de cada um significa, em virtude de seus diferentes ganhos. Assim, este artigo analisa experimentalmente o desempenho destes algoritmos em termos de tempo de execução, de espaço de memória e de qualidade da solução. Ademais, há o intuito de relacionar os resultados dessa medições com as características gerais das instâncias de teste.

O restante do artigo esta organizado da seguinte maneira: a seção 2, Conceitos Básicos, busca elucidar elementos simples do estudo, discorrendo sobre o problema da mochila e as estratégias utilizadas por cada algoritmo, a seção 3, Análise, procura explicar cada método, de forma a explicitar seu comportamento assintótico e sua corretude, a seção 4, Experimentos, apresenta os resultados obtidos dos experimentos, descrevendo métricas de cada teste, a seção 5, Discussão, visa criar inferências a partir dos desempenhos observação, havendo o intuito de comparar os compromissos de cada algoritmo, e a seção 6, Conclusão, resume o conteúdo do artigo e as ideias produzidas neste, o que tem o propósito de demonstrar as percepções criadas pela análise experimental.

2. Conceitos Básicos

Abordamos inicialmente alguns conceitos importantes para essa análise e aqui será explicado melhor sobre.

2.1. Problema da Mochila

O problema da Mochila é um problema otimização combinatória que consiste em selecionar N itens com valores e pesos específicos, de modo a maximizar o valor total carregado em uma mochila, sem ultrapassar sua capacidade limite W. Ou seja, há uma capacidade de peso da mochila, W, e um conjunto de N itens, cada um com um peso, w, e valor, v, tal que $v_i, w_i \in \mathbb{R}^+,\ W > 0$, desejando-se escolher um subconjunto de itens tal que $\sum_{i \in S} w_i \leq W$ e $\max \sum_{i \in S} v_i$. Por fim, a versão binária, tradada no artigo, impõe o limite de que cada item pode ser escolhido uma única vez. [Reilly 2004]

2.2. *Branch-and-Bound*

Branch-and-Bound é um tipo de algoritmo para encontrar soluções ótimas (solução exata) para problemas de otimização (escolha de uma melhor solução dentre várias outras possíveis). Consiste em enumerar sistemicamente todos os candidatos para solução, eliminando aqueles que sejam instâncias inviáveis ou não possuam resultados tão bons quanto os demais, junto de seus derivados (nós filhos). [Clausen 1999]

2.3. Programação dinâmica

Programação dinâmica é um método de construção de algoritmos para resolução de problemas computacionais no qual a solução ótima pode ser computada a partir da memorização de soluções ótimas parciais. Isso evita recálculos desnecessários, mas consome mais memória ao longo da execução. [Thomas H. Cormen 2022]

2.4. Algoritmos Aproximativos

São algoritmos usados para encontrar soluções aproximadas em problemas de otimização. Normalmente estão associados à ideia de problemas *NP* (que podem ser resolvidos em uma máquina de Turing não determinística em tempo polinomial) já que encontrar suas soluções em máquinas reais, determinísticas, no geral, requerem tempo exponencial. Mas também são utilizados em problemas *P* (que podem ser resolvidos em tempo polinomial por uma máquina de Turing determinística) para buscar soluções mais rápidas. [Kann 1992] A ideia do aproximativo se dá por uma constante *C* que diz o quão ruim é o pior resultado possível retornado pelo algoritmo. Logo, quando falamos sobre um algoritmo 2-aproximativo, temos que o resultado é no máximo 2 vezes pior que o resultado ótimo.

2.4.1. FPTAS

O FPTAS é um tipo de algoritmo aproximativo que, a partir de uma instância do problema e um parâmetro $\epsilon > 0$, retorna uma solução com valor pelo menos tão boa quanto $(1 - \epsilon)$ o valor ótimo. Ou seja, para problemas de maximização, o FPTAS garante que o valor da solução encontrada é pelo menos $(1 - \epsilon)$ vezes o valor ótimo, e em problemas de minimização, a solução encontrada é no máximo $(1 + \epsilon)$ vezes o valor ótimo. [Woeginger 2000]

2.4.2. Estratégia Gulosa

A estratégia gulosa é uma técnica utilizada por algoritmos que tentam construir uma solução ótima em um problema fazendo escolhas ótimas localmente. Por exemplo no problema de encontrar a maior soma nos pesos das arestas de um grafo valorado, a estratégia gulosa vai, a partir da raiz, escolher sempre os vértices adjacentes com maior valor. Na solução de alguns problemas consegue chegar no resultado ótimo, no entanto em muitos dos problemas *P* e *NP* essa estratégia se torna mais relevante do ponto de vista aproximativo. [Thomas H. Cormen 2022]

3. Implementação e Análise

Os algoritmos responsáveis por resolver o *Problema da Mochila Binário* foram escritos em *C++11*, e utilizam um tipo abstrato de dados (TAD), *Item*, que armazena dois números de ponto flutuante: peso *w* e valor *v*. Estes algoritmos, que foram implementados em uma classe *AlgoritmosMochila*, recebem um vetor de *Item* de tamanho N, o qual corresponde à lista de itens disponíveis para uma instância do problema, e um inteiro W, correspondente à capacidade máxima da mochila.

3.1. *Branch-and-Bound*

A abordagem de *Branch-and-Bound* explora iterativamente uma árvore de enumeração ordenada por níveis, do maior ao menor valor relativo $\frac{v}{w}$, usando limites estimados para podar ramos que não podem melhorar a melhor solução atual. A estimativa de custo escolhida, por ser cálculável em tempo constante, foi o maior valor relativo $\frac{v}{w}$ não utilizado até o nível atual, multiplicado pelo espaço restante na mochila. No pior caso, visita todos os 2^n nós da árvore, obtendo complexidade temporal exponencial, de $O(2^n)$. Como foi

escolhida a estratégia *Best First Search*, a fim de criar podas mais poderosas na operação de *bound* é utilizada uma fila de prioridade, que, no pior caso, pode armazenar todos os nós criados, obtendo também uma complexidade de espaço exponencial, de $O(2^n)$. Sua solução, quando encontrada, é sempre a ótima. Ademais, é válido mencionar que os nós são representados por um TAD *Node*, que armazena o nível atual da árvore, o peso e valor total dos itens na mochila, além do limite superior do valor total possível de se obter a partir do nó - com um comparador baseado no valor total.

3.2. FPTAS 2-aproximativo

A implementação do FPTAS escolhida transforma os valores do problema para inteiros aproximados e aplica programação dinâmica sobre a soma total de valores aproximados. Por possuir um laço duplo que calcula o valor total, com ϵ constante e igual a 0.5, sua complexidade de tempo se torna $O(n^3)$. Por armazenar uma tabela e um vetor de tamanhos proporcionais ao valor total, possui complexidade de espaço $O(n^2)$. O fator de aproximação igual a 2 não garante uma solução ótima, mas garante sua proximidade com a solução ótima por no máximo uma margem conhecida. Além disso, é imperativo citar que o algoritmo usa três vetores como estrutura de dados, um para guardar os valores transformados dos itens, outro para guardar a linha da tabela de DP para a iteração, e um último para manter quais itens foram escolhidos durante a execução da parte de programação dinâmica, o que permite encontrar um solução sobre os valores reais do problema.

Para demonstrar que o algoritmo possui uma garantia de qualidade de, no máximo, duas vezes pior que o resultado ótimo, isto é, que o algoritmo é 2-aproximativo, considere n o número de itens disponíveis, v_i e w_i o valor e peso do item i, ϵ o parâmetro de erro do FPTAS, μ o fator de escala, dado por $\frac{V_{\max}\epsilon}{n}$, sendo $V_{\max}$ o maior v_i, $\hat{v}_i$ o valor escalonado usado na programação dinâmica, o qual é obtido por $\left\lfloor \frac{v_i}{\mu} \right\rfloor$, e $\hat{S}$ o subconjunto de itens escolhidos por esta DP. Para cada item i, $v_i - \mu < \hat{v}_i\mu \leq v_i$, pois $\lfloor x \rfloor \geq x - 1$ e pelo fato do piso nunca exceder o valor original. Nesse sentido, tomando OPT e $O\hat{P}T$ como, respectivamente, os valores ótimos do problema original e do escalonado, é possível somar sobre o conjunto ótimo do problema original S^* para obter: $\mu O\hat{P}T = \mu \sum_{i \in S^*} \hat{v}_i \geq \sum_{i \in S^*} v_i - |S^*|\mu \geq OPT - n\mu$ **(1)**, visto que $|S^*| \leq n$. Ao substituir pela definição de μ, a expressão se torna $\mu O\hat{P}T \geq OPT - n\frac{V_{\max}\epsilon}{n} = OPT - V_{\max}\epsilon$. Como $V_{\max} \leq OPT$, segue que $\mu O\hat{P}T \geq (1 - \epsilon)OPT$ **(2)**. Ademais, como a DP usa o peso mínimo para cada valor escalonado e escolhe o maior valor $\hat{V}$ que respeite o limite da mochila W, $\hat{V} = \sum_{i \in \hat{S}} \hat{v}_i \geq O\hat{P}T$, sendo $\hat{S}$ ótimo para o problema escalonado. Nesse contexto, multiplicando a equação **(2)** por $\frac{\hat{V}}{O\hat{P}T} \geq 1 : \mu\hat{V} \geq \mu O\hat{P}T \geq (1 - \epsilon)OPT$. E, por fim, aplicando a equação **(1)** à solução de $\hat{S}$, $valor(\hat{S}) = \sum_{i \in \hat{S}} \geq \mu\hat{V} \geq (1 - \epsilon)OPT$. Assim, ao escolher $\epsilon = 0.5$ e substituir na última inequação, valor$(\hat{S}) \geq (1 - 0.5)OPT$, ou seja, $\frac{OPT}{\text{valor}(\hat{S})} \leq 2$, o que caracteriza o algoritmo como 2-aproximativo. [Kleinberg and Éva Tardos 2005]

3.3. Estratégia Gulosa

A estratégia gulosa 2-aproximativa escolhe entre o item de maior valor e uma solução incremental obtida a partir da seleção dos itens com maiores valores iterativos $\frac{v}{w}$ que não ultrapassa a capacidade máxima. Possui complexidade de tempo $O(n \log n)$, dominada pela ordenação dos itens pelos valores relativos $\frac{v}{w}$ de forma decrescente. Por armazenar apenas o vetor de itens ordenados, possui complexidade de espaço O(n).

Para a demonstração do fator de aproximação da estratégia gulosa, considere W a capacidade da mochila, G o conjunto de itens escolhidos pela solução gulosa - isto é, a sequência que coube na mochila -, V_G e W_G os somatórios dos valores e pesos dos itens em G, respectivamente, B o valor do melhor item isolado que cabe, $B = \max\{v_i | w_i \leq W\}$, ALG o valor retornado pelo algoritmo, $ALG = \max\{V_G, B\}$, e o fato dos itens estarem ordenados com base na razão $\frac{\text{valor}}{\text{peso}}$. Nesse sentido, a demonstração pode ser dividida em dois casos $W_G \geq \frac{W}{2}$ e $W_G < \frac{W}{2}$. No primeiro caso, todos os itens não escolhidos por G possuem uma razão menor ou igual a $\frac{V_G}{W_G}$, pois os itens já estão ordenados, o que garante que a razão média da solução ótima não pode ser maior que a de G, $\frac{OPT}{W} \leq \frac{V_G}{W_G}$, e, por conseguinte, $V_G \geq \frac{W_G}{W} OPT \geq \frac{W/2}{W} OPT = \frac{1}{2} OPT$. Portanto, $ALG \geq V_G \geq \frac{1}{2} OPT$. No segundo caso, o loop da solução gulosa para, pois o próximo item j não cabia na mochila, $w_j \geq W - W_G \geq \frac{W}{2}$, ou seja, qualquer item não escolhido por G tem peso maior que $\frac{W}{2}$. Sob tal ótica, nenhum par desses itens cabe na mochila e, consequentemente, o ótimo consiste em G mais, no máximo, um item adicional x que caiba sozinho: $OPT \leq V_G + v_x \leq V_G + B$. Se $V_G \geq \frac{1}{2} OPT$, a condição de aproximação já é satisfeita, mas, se $V_G < \frac{1}{2} OPT$, então $B \geq OPT - V_G > \frac{1}{2} OPT$. Assim, em todos os cenários $ALG = \max\{V_G, B\} \geq \frac{1}{2} OPT$, o que demonstra que o algoritmo é 2-aproximativo. [Chekuri 2009]

4. Experimentos

Os experimentos foram realizados sobre três óticas distintas: tempo de execução, espaço de memória e qualidade da solução, com o intuito de proporcionar detalhes sobre o desempenho dos algoritmos analisados. Nesse sentido, os dados, utilizados para permitir um panorama geral sobre as diversas características dos métodos, foram retirados de sites disponibilizados publicamente [Ortega 2025] [sc0v0ne 2024], que compilam conjuntos de testes usufruídos em um artigo relacionado ao tema [Pisinger 2005]. Eles foram organizados em baterias de testes de baixa e alta complexidade, a qual é determinada pela quantidade de itens disponíveis e pela capacidade da mochila.

A partir do tratamento desses dados, houve a execução de cada um dos testes para cada algoritmo, dentro de um limite de 30 minutos, isto é, os resultados das execuções cujo tempo ultrapassou esse limite foram considerados *Not Available*. As métricas dos três aspectos observados foram armazenadas em um arquivo *.csv*, o que permitiu a criação dos gráficos mostrados abaixo.

Para o tratamento dos dados e resultados foi utilizado *Python3*, e para as medições de tempo e espaço, houve o emprego da biblioteca em *sys/resource.h*. Ademais, para cada instância, além do número de itens N e a capacidade da mochila W, também foram coletadas informações sobre o peso máximo *wmax*, peso médio *wavg*, valor máximo *vmax* e valor médio *vavg*. Os testes foram executados em um ambiente padrão do WSL2 (*Windows Subsystem for Linux*).

4.1. Tempo de Execução

A extração do tempo de execução é baseada na soma do tempo decorrido, desde o começo da chamada do algoritmo até o final, para o sistema e para o usuário.

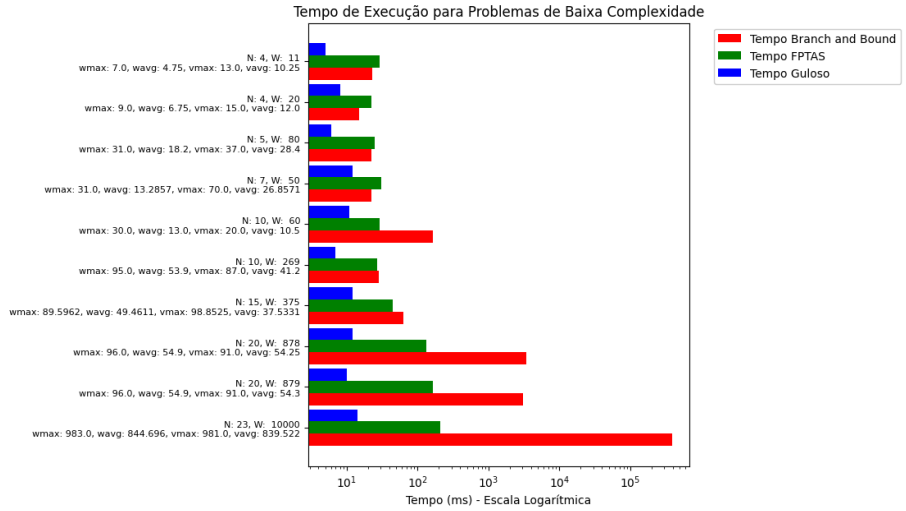

Figura 1. Tempo de Execução para Problemas de Baixa Complexidade

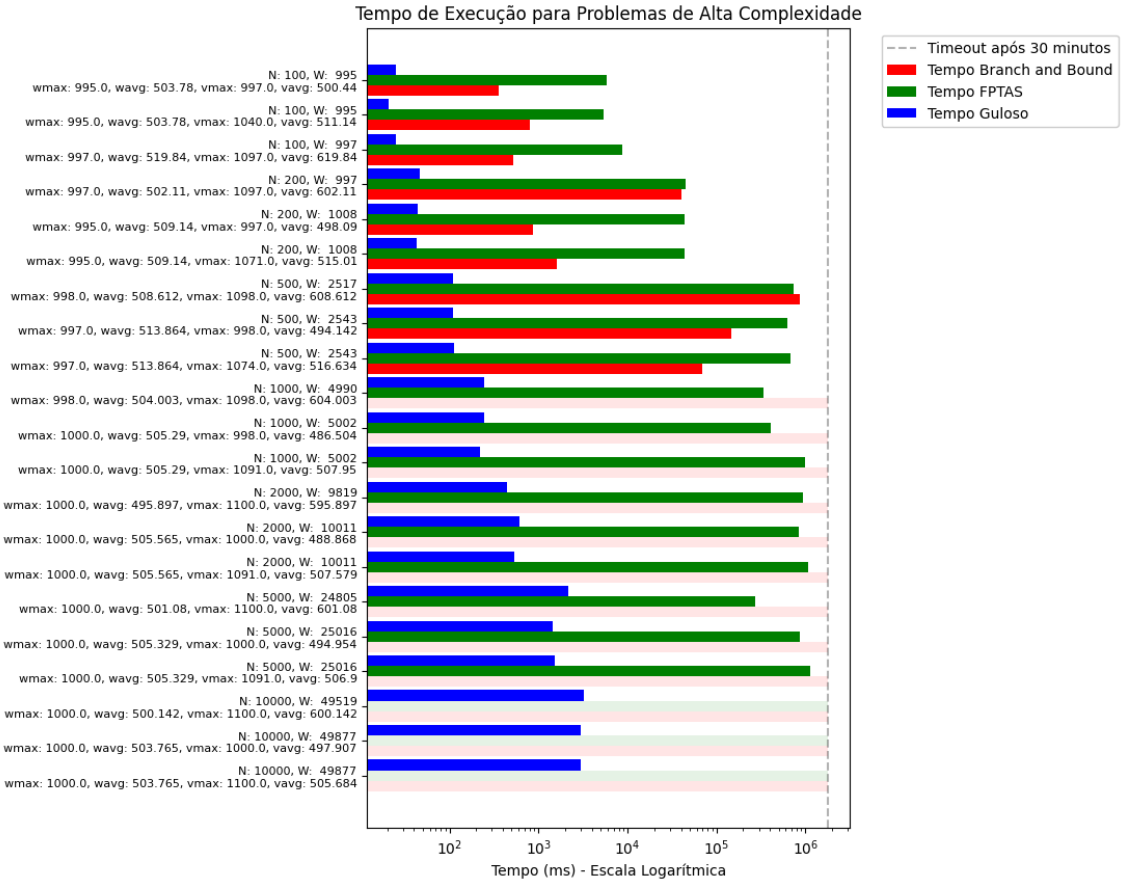

Figura 2. Tempo de Execução para Problemas de Alta Complexidade

4.2. Espaço de Memória

A extração da memória de espaço é feita obtendo o pico de uso da memória física principal para cada execução.

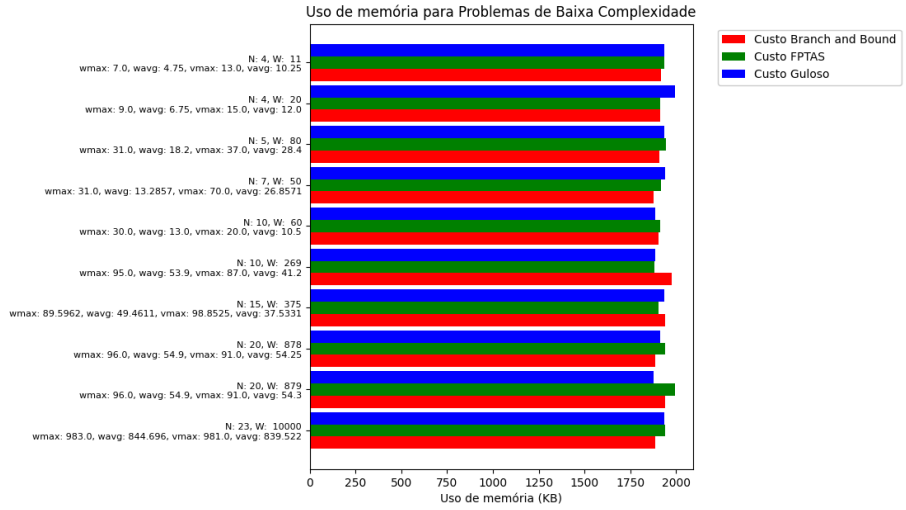

Figura 3. Uso de memória para Problemas de Baixa Complexidade

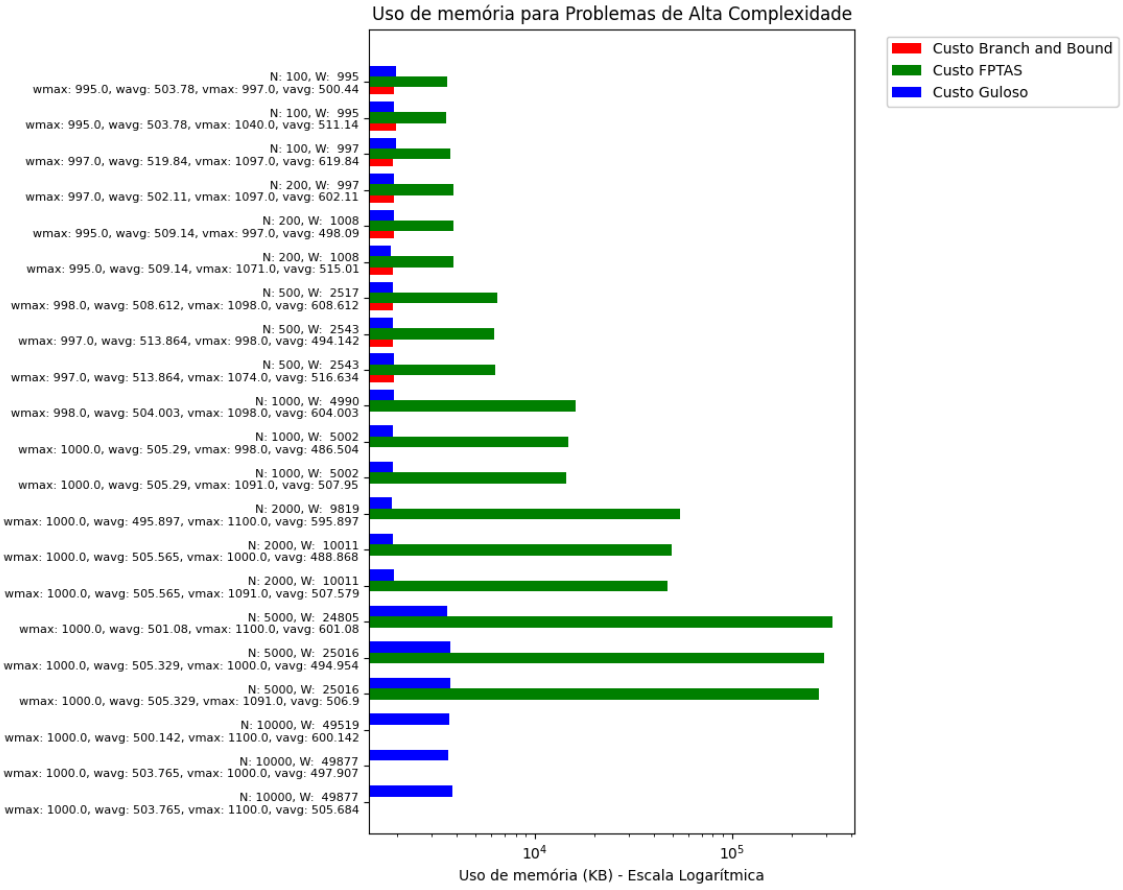

Figura 4. Uso de memória para Problemas de Alta Complexidade

4.3. Qualidade da Solução

A qualidade da solução é baseada na comparação da resposta dada pelos algoritmos com a solução ótima.

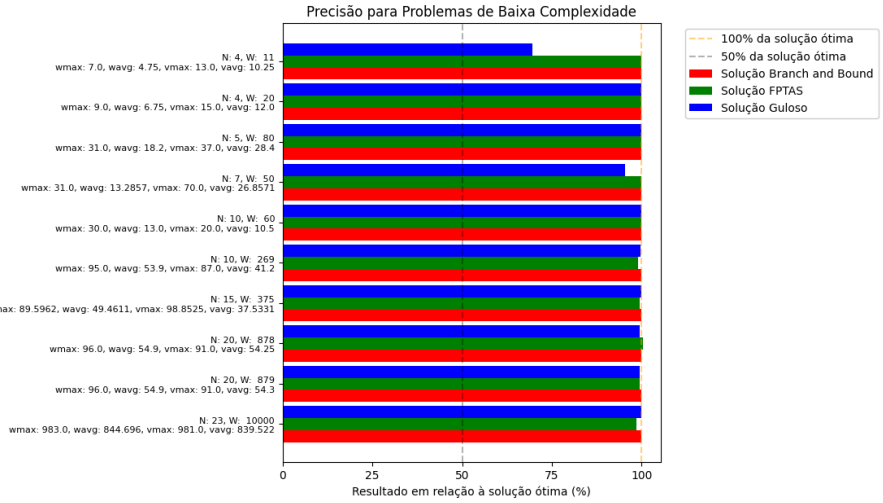

Figura 5. Precisão para problemas de Baixa Complexidade

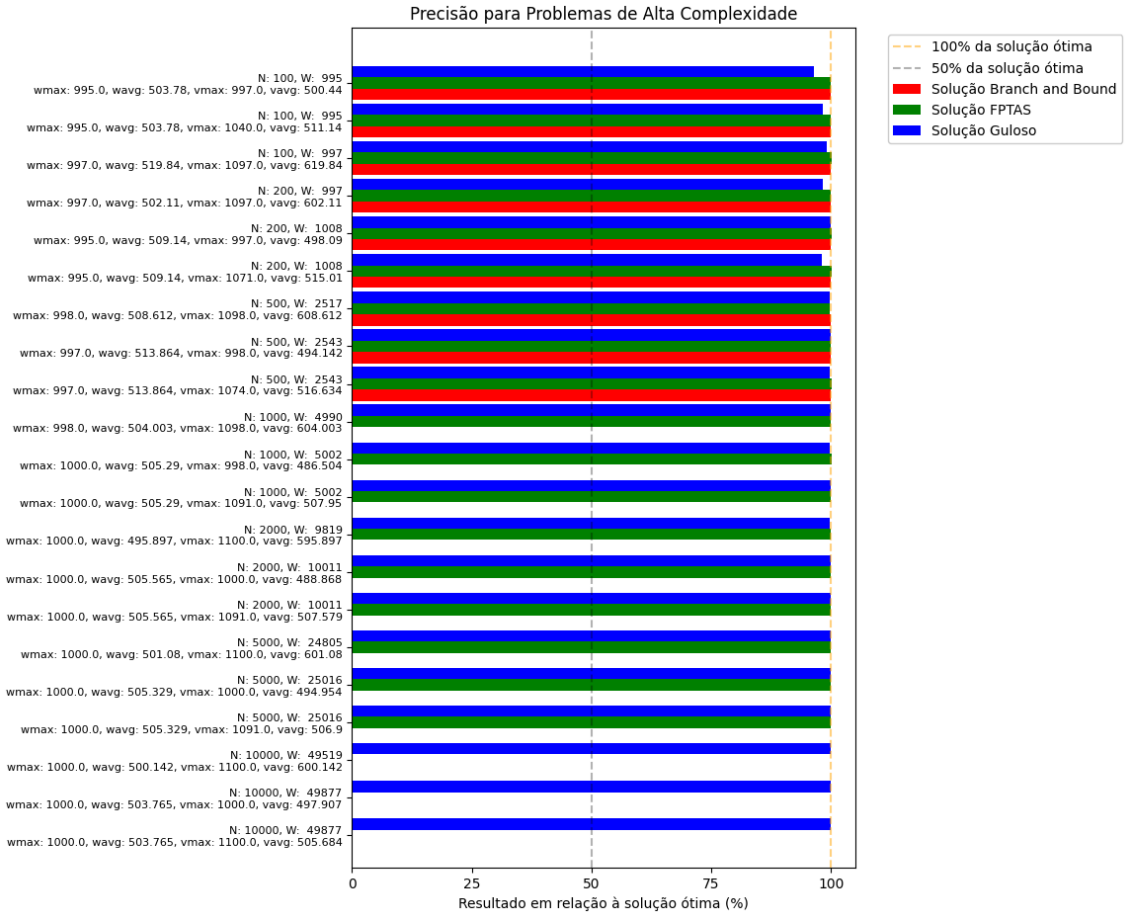

Figura 6. Precisão para problemas de Alta Complexidade

5. Discussão dos resultados

Ao analisar os testes, é possível notar que há um *tradeoff* entre optimalidade e tempo de processamento e espaço gasto pelos algoritmos. Ou seja, quanto mais próximo do resultado correto, mais tempo é necessário para executar o algoritmo.

Ao analisar a complexidade temporal, é evidente que, para instâncias maiores o *Branch-and-Bound* é o algoritmo que mais leva tempo para ser executado, uma vez que ele é o único que busca resultados ótimos. O maior destaque entre os aproximativos é o algoritmo do tipo Guloso que obteve uma vantagem temporal consideravelmente maior, quando comparada com o FPTAS, devido ao seu custo assintótico menor. Assim, em conjuntos no qual o *Branch-and-Bound* e o FPTAS não podem ser executados em tempo hábil, o guloso consegue reproduzir o resultado aproximativo em tempos, de fato, muito melhores. Além disso, foi possível notar que em algumas instâncias, principalmente naquelas nas quais a proporção $\frac{W}{wavg}$ é maior, o *Branch-and-Bound* pôde podar mais ramos da árvore, obtendo um tempo de execução menor.

Em uma análise espacial, é possível verificar que, nos casos em que os algoritmos executaram dentro do tempo hábil, o *Branch-and-Bound* obteve um gasto relativamente baixo de memória, pois nunca chegou a construir a árvore de decisão inteira, pela operação de *bound*. Já entre os aproximativos, o guloso se demonstrou superior, já que o FPTAS ainda precisa sempre construir a matriz da programação dinâmica inteira, consumindo um espaço de memória muito maior do que os outros algoritmos.

Ao analisar a precisão dos algoritmos nos testes realizados, o *Branch-and-Bound* sempre busca o ótimo, então encontra, sem surpresas a solução correta. Entre os 2-aproximativos, na maior parte dos casos, o FPTAS compensa o alto custo assintótico temporal e espacial na sua precisão, que se manteve longe de seu pior caso. Já no guloso, é notável que as soluções são, no geral, mais distantes da ótima do que as do FPTAS, mas ainda assim se mantendo proporcionalmente mais próximas da ótima do que do pior caso, nas baterias de testes de alta complexidade.

Em síntese, foi observado que os aproximativos estão bem próximos ou retornam o resultado ótimo na maioria dos casos, ou seja, no final, o *tradeoff* realizado por ambos é compensado de alguma forma nas instâncias maiores. Nesse sentido, as análises mostraram que cada um dos aproximativos teve seu erro relativo reduzido à medida que o problema cresceu, abrindo espaço para aplicação desses métodos em diversos casos de uso.

6. Conclusão

Com isso, através de experimentos de baixa e larga escala, foi possível avaliar as diferentes qualidades e pontos fracos dos algoritmos analisados, e verificar que, no geral, os aproximativos, principalmente o guloso, compensam para problemas grandes, nos quais a obtenção da solução ótima não é crítica, visto que o *tradeoff* entre precisão e tempo é compensatório. Além disso, se o custo espacial for o principal limitante, o uso do guloso é o mais indicado nessa situação, no entanto, quando não houver esse limitante e o tempo for viável, o *Branch-and-Bound* é o mais indicado por sua precisão total. Ademais, é válido citar que, apesar do FPTAS apresentar um desempenho, em termos de tempo e de memória, pior que a alternativa gulosa, sua vantagem está presente na possibilidade de alterar seu fator de aproximação sob-demanda, enquanto o guloso tem uma precisão fixa, sendo preciso criar outros algoritmos e outras lógicas para obter outros fatores.

Referências

Chekuri, C. (2009). *Approximation Algorithms*.

Clausen, J. (1999). *Branch and Bound Algorithms - Principles and Examples*.

Kann, V. (1992). *On the Approximability of NP-complete Optimization Problems*.

Kleinberg, J. and Éva Tardos (2005). *Algorithm Design*. Pearson, 1 edition.

Ortega, J. (2025). *Instances of 0/1 Knapsack Problem*.

Pisinger, D. (2005). *Where are the hard knapsack problems?*

Reilly, E. D. (2004). *Concise encyclopedia of computer science*.

sc0v0ne (2024). *Dataset Large Scale - 0/1 knapsack problems*.

Thomas H. Cormen, Charles E. Leiserson, R. L. R. C. S. (2022). *Introduction To Algorithms The MIT Press*. The MIT Press, 4 edition.

Woeginger, G. J. (2000). *When Does a Dynamic Programming Formulation Guarantee the Existence of a Fully Polynomial Time Approximation Scheme (FPTAS)?*