Relatório trabalho prático 2 - Algoritmos 2

Lucas Affonso Pires[1], **Mateus Faria Zaparoli Monteiro**[1]

[1]Departamento de Ciência da Computação – UFMG

{mateuszaparoli7, lap110303}@ufmg.br

Abstract. *The binary knapsack problem is a classic problem in the area of algorithms and optimizations. There are several algorithms to solve this problem. In this work, we will address three of these algorithms: the branch-and-bound [Vimieiro 2025a] algorithm, an FPTAS-type algorithm [Vimieiro 2025d], and a 2-approximate algorithm [Vimieiro 2025b] [Vimieiro 2025c]. We will study the advantages and disadvantages of each one, in relation to time, space, and quality of the solution. It can be seen, for example, from the tests performed, that for small instances that favor the cuts of the branch-and-bound it can be faster than the FPTAS even with larger ϵ. However, the 2-approximate algorithm is still faster because it is a non-exact algorithm and the branch-and-bound is exact, showing that for these instances, approximation algorithms can present better solutions in time without greatly compromising quality.*

Resumo. *O problema da mochila binário é um clássico problema na área de algoritmos e otimizações. Existem diversos algoritmos para resolver esse problema, nesse trabalho abordaremos três destes algoritmos, serão eles o algoritmo de branch-and-bound [Vimieiro 2025a] , um algoritmo do tipo FPTAS [Vimieiro 2025d] e um algoritmo 2-aproximativo [Vimieiro 2025b] [Vimieiro 2025c]. Estudaremos as vantagens e desvantagens de cada um, em relação a tempo, espaço e qualidade da solução, nota-se, pelos testes feitos, por exemplo, que pra instâncias pequenas e que favoreçam os cortes do branch-and-bound ele pode ser mais rápido que o FPTAS mesmo este com ϵ maiores, porém o 2-aproximativo ainda se mostre mais rápido por ser um algoritmo não exato e o branch-and-bound seja exato, mostrando que para essas instâncias que algoritmos aproximativos podem apresentar soluções melhores em tempo sem prejudicar muito a qualidade.*

1. Introdução

Algoritmos que buscam soluções para o problema da mochila são amplamente estudados, existem diversas abordagens para solucionar esse problema, como algoritmos aproximativos [Wikipédia 2025], que garantem que sua solução está no máximo x vezes longe da solução ótima, no caso de algoritmos x-aproximativos, por exemplo. Existem ainda os algoritmos que implementam o *branch-and-bound*, que divide a resolução em ramos de uma árvore *branch* cada um com uma possibilidade de valor para as variáveis na solução final, e poda os ramos (*bound*) com base em quão promissor um ramo específico é, dada uma solução já encontrada. Existem ainda a classe dos algoritmos PTAS e FPTAS, os PTAS, polynomial-time approximation scheme, são algoritmos que encontram as soluções em tempo polinomial com o tamanho da entrada enquanto os FPTAS, fully polynomial-time

approximation scheme, são algoritmos que garantem uma solução em tempo polinomial não só com o tamanho da entrada mas também polinomial em $\frac{1}{\epsilon}$, o ϵ em ambos os casos representa o quão longe a solução que encontramos com esses algoritmos está da solução ideal, ou seja, para um dado ϵ a solução encontrada é, no máximo, $1 - \epsilon$ vezes menor que a solução ideal, ou $1 + \epsilon$ vezes maior que a solução ótima.

No presente trabalho foram implementados 3 algoritmos, o primeiro um algoritmo de *branch and bound* para encontrar soluções do problema da mochila binária, além desse primeiro também foram implementados um algoritmo FPTAS para mesmo problema e ainda um algoritmo 2-aproximativo ainda para o mesmo problema.

O trabalho visa implementar e testar, para diferentes instâncias, os diferentes algoritmos e como isso entender como cada um funciona e situações de uso de cada um, por exemplo, para cada carga de trabalho e necessidade de acurácia e tempo de resposta, qual dos algoritmos é mais adequado.

2. Materiais e Métodos

Nessa seção serão detalhadas as abordagens utilizadas para a realização do trabalho. Primeiramente, a respeito da implementação do algoritmo de *branch and bound*, utilizou-se como base a implementação vista em sala desse algoritmo, primeiro criou-se uma classe que define um item, composto de índice, peso, valor e razão (entre valor e peso). Após a definição dessa classe começamos o algoritmo, primeiramente ordenamos os itens em ordem decrescente da razão $\frac{valor}{peso}$, para poder calcular corretamente os limites (bounds) na execução do algoritmo, após isso definimos uma função chamada limite que calcula o limite superior que podemos obter a partir de um dado nó da árvore, se o peso no nó já ultrapassou à capacidade da mochila retornamos 0 nessa função, depois adicionamos os itens inteiros a possíveis a partir do nó atual e depois adicionamos a fração que restar do último item restante, retornando assim um limite superior para aquele dado ramo da árvore. Após isso, representamos o nó como uma tupla em que o primeiro elemento é o limite superior negado para assim podermos usar o heapq como uma fila de prioridade máxima já que no python o heapq é implementado como um min-heap, negando o limite superior transformamos ele em um max-heap, em seguida realizamos uma busca que chamamos de best-first em que exploramos primeiros os nós mais promissores e podamos os galhos que têm uma solução máxima possível menor que a solução atual, se o nó explorado tiver um máximo possível maior que o atual adicionamos mais dois nós à árvore, um nó com o próximo item, e um nó sem o próximo item, ao final teremos a solução encontrada pelo algoritmo e retornamos o valor dela bem como os itens que a compõem.

Essa solução tem complexidades exponenciais no pior caso, pois, nesse caso, o algoritmo acaba tendo que passar por todos os nós e percorrer todos os ramos da árvore completamente, o que leva a uma complexidade temporal de $O(2^n * log 2^n$ que seria o custo de gerar todos os subconjuntos possíveis de elementos vezes o custo de inseri-los e removê-los do heapq, tal custo é logaritmo. Além disso, a complexidade espacial dessa solução no pior caso também é exponencial, sendo de $O(2^n)$ que seria o custo de armazenar todos os nós de subconjuntos de elementos possíveis. Ainda assim, poderemos ver ao longo desse próprio trabalho que na prática, no caso médio, e em grande parte dos casos as podas e a execução desse algoritmo são rápidas não sendo exponencial em tempo nem em espaço, e, para diversas instâncias sendo melhor que os algoritmos que vamos

apresentar na sequência.

Em seguida, foi implementado o algoritmo FPTAS para o problema da mochila binário, primeiramente iniciamos os parâmetros n, v_max e mi do FPTAS, em seguida, usando o mi escalonamos os valores para assim, seguindo uma das principais funções do FTPAS, diminuir a complexidade do problema, aí então criamos e preenchemos uma tabela, a fim de fazer uso da programação dinâmica, o tamanho da tabela, ou array, é $\mathbf{V + 1}$ onde V é a soma dos valores dos itens depois do escalonamento, para preenchê-la por cada item percorremos os possíveis valores de v de forma decrescente e verificamos se é possível atingir v com o item de índice i, se sim adicionamos o valor da posição v da tabela com o menor valor possível. Por fim, percorremos a tabela montada do fim para o início para procurar o maior valor possível cujo peso total ainda respeite a capacidade da mochila e multiplicamos por mi para ter o valor aproximado original da solução.

Analisando a complexidade temporal desse algoritmo temos que para cada item da mochila percorremos as V posições na tabela, como o tamanho de V é $O(\frac{n}{\epsilon})$, temos que a complexidade temporal desse algoritmo é $O(\frac{n}{\epsilon})$. Em relação, por sua vez, à complexidade espacial, guardamos, durante a execução do algoritmo o vetor da programação dinâmica que tem tamanho proporcional a $\frac{1}{\epsilon}$ vezes o número de itens. A compleixidade desse algoritmo em todos os casos ainda é polinomial, por isso, espera-se que ele seja muito mais eficiente que algoritmos exatos como o *branch-and-bound* em grandes instâncias.

Depois da implementação desses dois algoritmos foi implementado, então, um algoritmo 2-aproximativo para o problema da mochila binário, nele encontramos o maior elemento que, individualmente, cabe na mochila e também utilizamos uma abordagem gulosa que escolhe os itens com maior razão $\frac{peso}{valor}$. O algoritmo primeiro encontra o maior valor que sozinho cabe na mochila, depois ele segue para a abordagem gulosa primeiro ordenando em ordem decrescente os itens seguindo a razão $\frac{valor}{peso}$ após isso vamos adicionando os itens seguindo a ordenação até não caber mais itens na mochila, cuja capacidade não foi ultrapassada pelo peso dos itens escolhidos. Temos, então, o resultado da abordagem do maior item de valor que não ultrapassa o peso e, também, o resultado da abordagem, aí então escolhemos o maior entre eles. Esse algoritmo é 2-aproximativo, vamos provar usando o fato de que $OPT\_f >= OPT$, ou seja, a solução ótima do problema da mochila fracionário é maior ou igual à solução ótima do problema inteiro, então provamos que $ALG >= \frac{1}{2} * OPT\_f$, no problema frarionário, após inserirmos todos os items que cabem inteiros colocamos uma fração do próximo item até encher a capacidade da mochila obtendo OPT_f, no caso do problema inteiro paramos no último item que cabe inteiramente. Então, provamos que esse valor é no mínimo metade da solução ótima fracionária, pois ou essa solução já é em si, maior que metade da solução ótima fracionária, e então a desigualdade já estaria satisfeita e a prova concluída, ou o valor do próximo item, que teve uma fração usada no problema fracionário é maior que metade da solução fracionária. Uma vez que nosso algoritmo compara a solução da escolha gulosa com o item de maior valor individual que cabe na mochila e escolhe o maior dentre as duas soluções, garantimos que temos uma solução que é no mínimo a metade da ótima fracionária, que por sua vez é maior que a solução ótima inteira provando assim que o algoritmo implementado é de fato 2-aproximativo.

Em relação às complexidades espaciais e temporais desse algoritmo temos que, em relação à complexidade temporal esse algoritmo é $O(n * logn$ que, no caso, é do-

minada pela ordenação dos itens decrescentemente pela razão $\frac{peso}{valor}$, as outras operações que fazemos são lineares, a complexidade espacial, por outro lado é $O(n)$ devido ao fato de guardarmos a razão antes mencionada para cada item, pelas complexidades apresentada e pela garantia da solução no mínimo metade da ótima para o caso de problemas de máximo, esse algortimo é bom mesmo em grandes instâncias e uma opção segura para diversas situações já que gastamos pouco tanto em memória como em tempo.

3. Resultados e Análises

Nesta seção, apresentamos e discutimos os resultados obtidos a partir da execução dos três algoritmos para o problema da mochila binária: Branch-and-Bound, FPTAS e o algoritmo 2-aproximativo. As métricas consideradas foram o valor da solução obtida, o tempo de execução (em segundos) e a memória utilizada (em kilobytes). Os testes foram realizados sobre instâncias de diferentes tamanhos, categorizadas como *low-dimensional* e *large-scale*.

3.1. Tempo de Execução

Os resultados mostraram que o algoritmo 2-aproximativo é significativamente mais rápido do que os demais, com tempos de execução na ordem de microssegundos para a maioria das instâncias. O Branch-and-Bound também apresentou excelente desempenho em grande parte dos casos, com tempos que raramente ultrapassavam 1 segundo de execução, com exceção, apenas, de instâncias como `f8_l-d_kp_23_10000`, onde o tempo aumentou consideravelmente, ultrapassando 27 segundos. O FPTAS apresentou os piores resultados para a maioria dos casos, com piora perceptível à medida que o tamanho das instâncias crescia, atingindo mais de 200 segundos nas instâncias mais complexas.

3.2. Memória Utilizada

Com relação ao uso de memória, o algoritmo 2-aproximativo novamente se destacou por seu baixo consumo, raramente ultrapassando 1KB. O Branch-and-Bound também se manteve eficiente na maioria das instâncias, mas apresentou picos de consumo significativo, como no caso da instância `f8_l-d_kp_23_10000`, onde ultrapassou 200 MB de memória, provavelmente por ser uma instância onde o tempo de execução para o algoritmo foi maior que o comum. Já o FPTAS demonstrou comportamento mais variável, com consumo moderado em instâncias pequenas e aumento considerável em instâncias maiores.

3.3. Valor da Solução

Como esperado, o algoritmo Branch-and-Bound sempre encontrou a solução ótima para todas as instâncias testadas, servindo como referência de qualidade (e, por essa razão, ele não foi analisado nas imagens que mostram o Desvio Percentual do Valor Ideal, pois seu valor sempre é ideal) . O FPTAS, com $\epsilon = 0.2$ ou $\epsilon = 0.5$ a depender da instância, apresentou soluções próximas ao ótimo, com perdas geralmente pequenas, variando entre 0.1% e 2% em relação ao valor ótimo. Já o algoritmo 2-aproximativo, embora mais rápido, teve, para a maioria dos casos, mais proximidade ao ideal que o FPTAS, porém, também obteve os maiores valores de desvio do ideal em alguns poucos casos. Tal característica mostra como o 2-aproximativo inclui riscos se a busca é um resultado próximo ao perfeito.

3.4. Imagens comparativas entre os Algoritmos

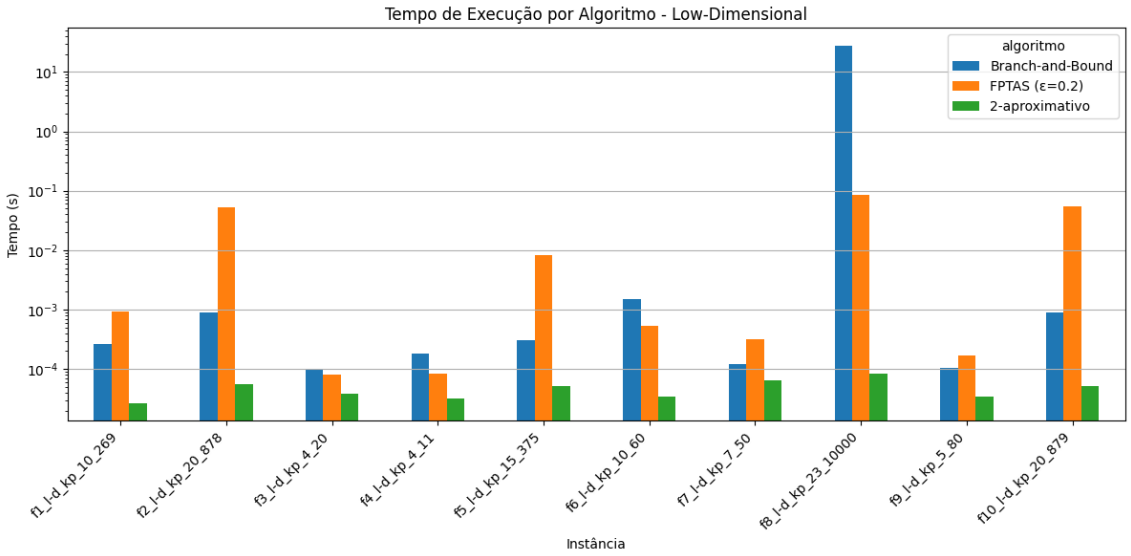

Figura 1. Testes Low-Dimensional quanto ao tempo

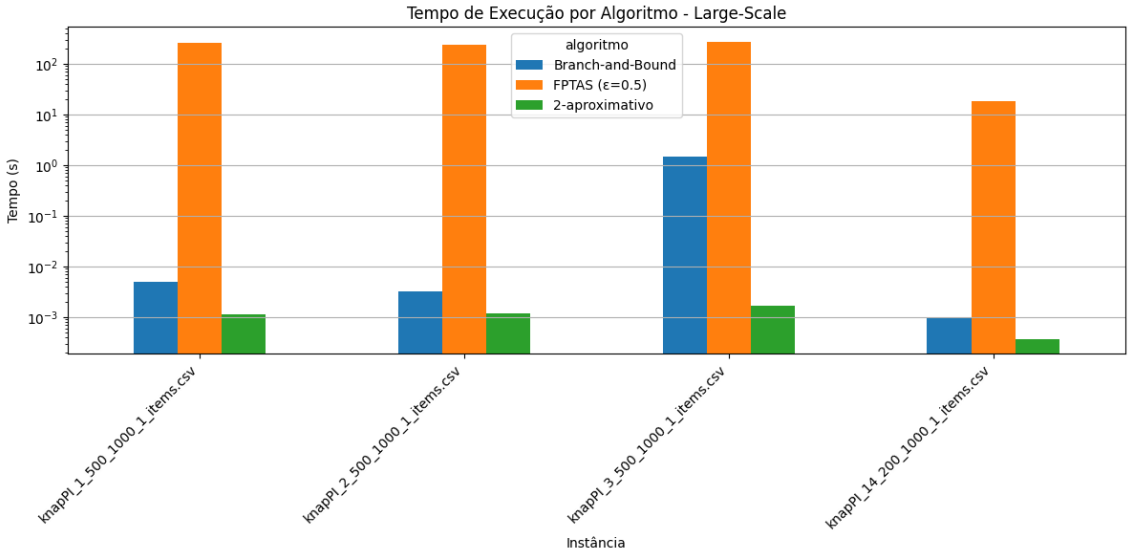

Figura 2. Testes Large-Scale quanto ao tempo

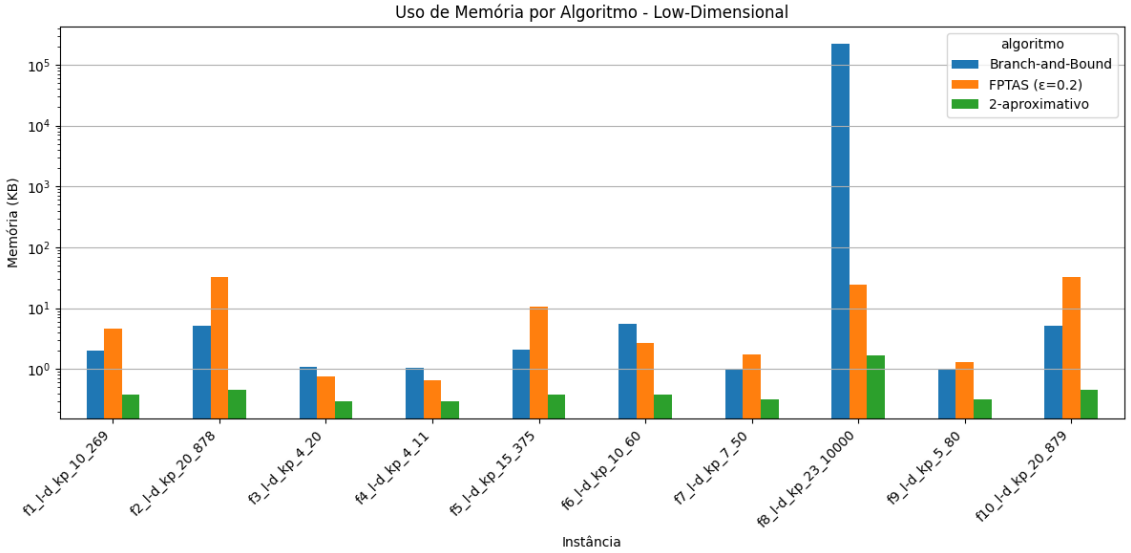

Figura 3. Testes Low-Dimensional quanto ao espaço

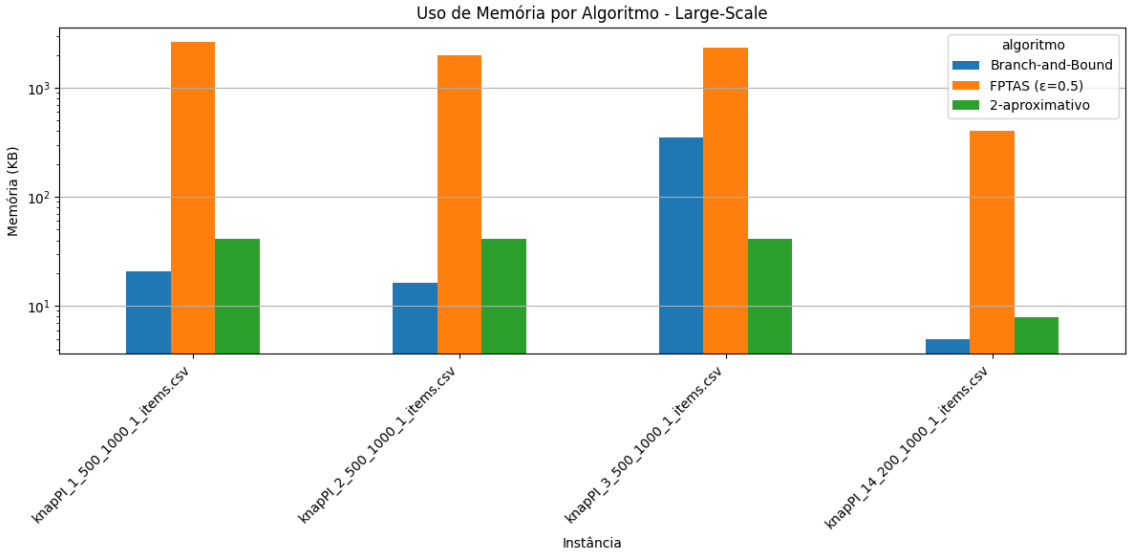

Figura 4. Testes Large-Scale quanto ao espaço

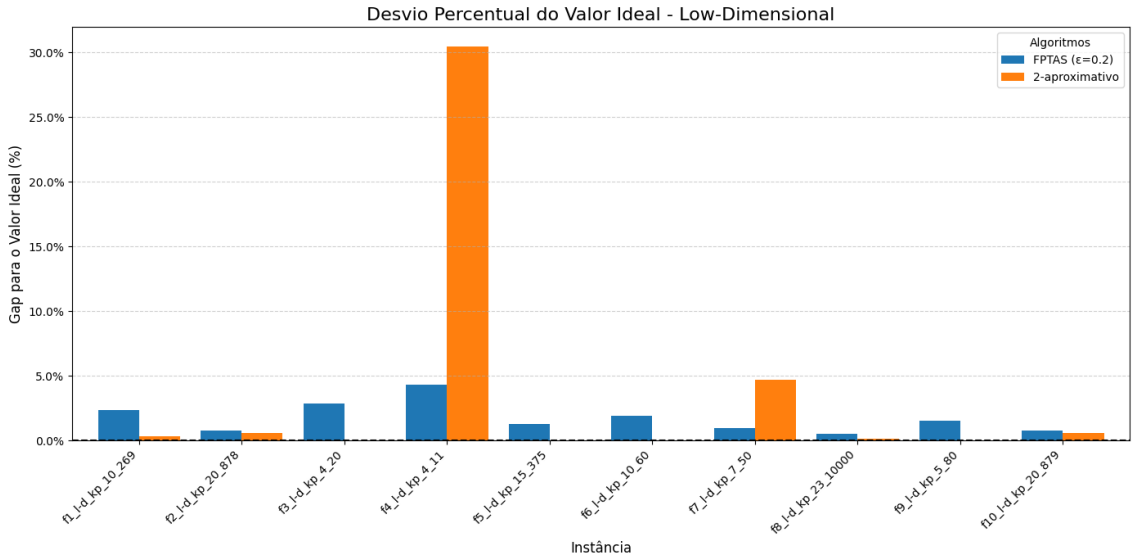

Figura 5. Testes Low-Dimensional quanto ao desvio percentual do valor ideal

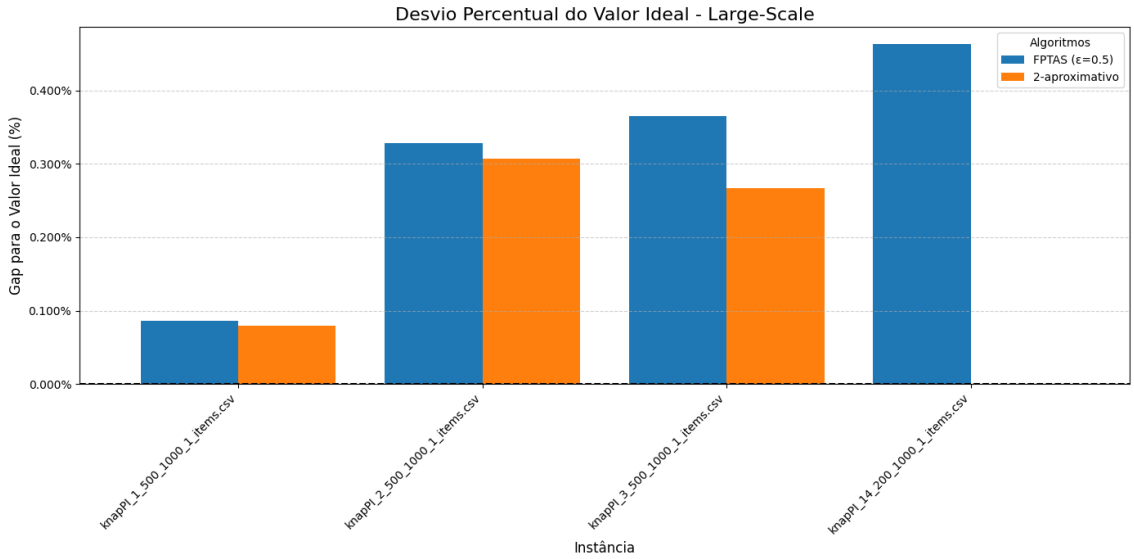

Figura 6. Testes Large-Scale quanto ao desvio percentual do valor ideal

3.5. Tabela de resultados

Instância	Tamanho	Algoritmo	Valor	Tempo (s)	Memória (KB)
f1_l-d_kp_10_269	low-dimensional	Branch-and-Bound	295.0	0.000264	1.99
f1_l-d_kp_10_269	low-dimensional	FPTAS (ϵ=0.2)	288.0	0.000929	4.69
f1_l-d_kp_10_269	low-dimensional	2-aproximativo	294.0	2.7e-05	0.38
f2_l-d_kp_20_878	low-dimensional	Branch-and-Bound	1024.0	0.000887	5.13
f2_l-d_kp_20_878	low-dimensional	FPTAS (ϵ=0.2)	1016.0	0.05348	32.89
f2_l-d_kp_20_878	low-dimensional	2-aproximativo	1018.0	5.5e-05	0.45
f3_l-d_kp_4_20	low-dimensional	Branch-and-Bound	35.0	0.000101	1.08
f3_l-d_kp_4_20	low-dimensional	FPTAS (ϵ=0.2)	34.0	8e-05	0.75
f3_l-d_kp_4_20	low-dimensional	2-aproximativo	35.0	3.9e-05	0.3
f4_l-d_kp_4_11	low-dimensional	Branch-and-Bound	23.0	0.000181	1.06
f4_l-d_kp_4_11	low-dimensional	FPTAS (ϵ=0.2)	22.0	8.4e-05	0.66
f4_l-d_kp_4_11	low-dimensional	2-aproximativo	16.0	3.2e-05	0.3
f5_l-d_kp_15_375	low-dimensional	Branch-and-Bound	481.07	0.000303	2.09
f5_l-d_kp_15_375	low-dimensional	FPTAS (ϵ=0.2)	475.0	0.008222	10.66
f5_l-d_kp_15_375	low-dimensional	2-aproximativo	481.07	5.2e-05	0.38
f6_l-d_kp_10_60	low-dimensional	Branch-and-Bound	52.0	0.001489	5.55
f6_l-d_kp_10_60	low-dimensional	FPTAS (ϵ=0.2)	51.0	0.000534	2.66
f6_l-d_kp_10_60	low-dimensional	2-aproximativo	52.0	3.5e-05	0.38
f7_l-d_kp_7_50	low-dimensional	Branch-and-Bound	107.0	0.000122	0.97
f7_l-d_kp_7_50	low-dimensional	FPTAS (ϵ=0.2)	106.0	0.000318	1.73
f7_l-d_kp_7_50	low-dimensional	2-aproximativo	102.0	6.5e-05	0.32
f8_l-d_kp_23_10000	low-dimensional	Branch-and-Bound	9767.0	27.506348	216999.84
f8_l-d_kp_23_10000	low-dimensional	FPTAS (ϵ=0.2)	9716.0	0.086557	24.72
f8_l-d_kp_23_10000	low-dimensional	2-aproximativo	9753.0	8.4e-05	1.7
f9_l-d_kp_5_80	low-dimensional	Branch-and-Bound	130.0	0.000103	1.02
f9_l-d_kp_5_80	low-dimensional	FPTAS (ϵ=0.2)	128.0	0.000167	1.31
f9_l-d_kp_5_80	low-dimensional	2-aproximativo	130.0	3.5e-05	0.32
f10_l-d_kp_20_879	low-dimensional	Branch-and-Bound	1025.0	0.000881	5.13
f10_l-d_kp_20_879	low-dimensional	FPTAS (ϵ=0.2)	1017.0	0.054672	32.95
f10_l-d_kp_20_879	low-dimensional	2-aproximativo	1019.0	5.2e-05	0.45
knapPI_1_500_1000_1_items.csv	large-scale	Branch-and-Bound	28857.0	0.004942	20.99
knapPI_1_500_1000_1_items.csv	large-scale	FPTAS (ϵ=0.5)	28832.0	261.622534	2620.0
knapPI_1_500_1000_1_items.csv	large-scale	2-aproximativo	28834.0	0.001133	41.46
knapPI_2_500_1000_1_items.csv	large-scale	Branch-and-Bound	4566.0	0.003276	16.44
knapPI_2_500_1000_1_items.csv	large-scale	FPTAS (ϵ=0.5)	4551.0	235.489236	1989.32
knapPI_2_500_1000_1_items.csv	large-scale	2-aproximativo	4552.0	0.001187	41.46
knapPI_3_500_1000_1_items.csv	large-scale	Branch-and-Bound	7117.0	1.486788	348.79
knapPI_3_500_1000_1_items.csv	large-scale	FPTAS (ϵ=0.5)	7091.0	273.786437	2324.55
knapPI_3_500_1000_1_items.csv	large-scale	2-aproximativo	7098.0	0.001665	41.46
knapPI_14_200_1000_1_items.csv	large-scale	Branch-and-Bound	5397.0	0.000965	4.99
knapPI_14_200_1000_1_items.csv	large-scale	FPTAS (ϵ=0.5)	5372.0	18.046668	401.2
knapPI_14_200_1000_1_items.csv	large-scale	2-aproximativo	5397.0	0.000374	7.88

Tabela 1. Resultado da execução dos algoritmos com tempo e memória

3.6. Análise Geral

A fim de gerar uma análise geral, como um guia de uso dos algoritmos, o que era, em certo ponto, uma das propostas do trabalho, nessa subseção trazemos uma breve análise de casos de usos e de particularidades de cada algoritmo.

Primeiramente, em relação ao algoritmo 2-aproximativo podemos observar pelos resultados acima demonstrados que este algoritmo é extremamente rápido e estável, seus tempos todos abaixo de 2ms para as instâncias testadas. O desvio padrão desse algoritmo é muito baixo o que indica também a consistência do algoritmo.

Segundamente, em relação ao FPTAS observamos que ele mostra-se, na implementação feita, com tempos baixos mas ainda maiores que o 2-aproximativos, porém mesmo com um ϵ pequeno ele ainda tem tempos absolutos baixos, o que mostra que esse algoritmo é uma boa balança entre qualidade da solução e tempo de execução.

Por fim, temos o algoritmo de branch and bound, pelos gráficos, tabelas, e análises acima podemos notar que esse algoritmo tem um média muito alta de execução em relação aos seus concorrentes nessa análise, podemos observar também que ele se mostra muito variável a depender dos tempos de execução dos algoritmos, apresentando tempos que chegam a mais de 27 segundos para algumas instâncias. Isso indica que esse algoritmo é o que mais pode sofrer ou ser prejudicado a depender das características da instância.

4. Discussão

O resultados apresentados no trabalho foram bem esclarecedores e de certa forma esperados, a implementação dos algoritmos e as análises dos resultados foram as etapas mais custosas do trabalho, para as implementações foram usadas como base as aulas do curso de Algoritmos II, nos slides vistos em sala e nos capítulos [Cormen et al. 2009] recomendados para leitura.

A respeito dos resultados obtidos podemos então resumidamente indicar usos para cada um deles. Se o desejo é um tempo mínimo de execução, para nossa implementação, o algoritmo escolhido para uso dever ser o 2-aproximativo, se for necessário um compromisso entre precisão e eficiência, deve-se escolher um ϵ pequeno e adequado usando o FPTAS para casos em que esse compromisso é importante. Se a qualidade da solução é crítica para o uso, então deve-se usar o algoritmo de *branch-and-bound*, embora seja necessário observar a natureza da solução evitando usar esse algortimo em instâncias muito grandes porque isso pode crescer muito o tempo e a memória usados por esse algortimo.

5. Conclusão

Concluímos, então, o trabalho com essas análises, o trabalho foi de grande aprendizado para os alunos para fixar conhecimentos obtidos na disciplina, e observar na prática quais algoritmos devem ser utilizados para situações específicas e reais e como implementar tais algoritmos.

Observamos então cada compromisso feito ao se utilizar cada algoritmo, como o 2-aproximativo para tempos melhores, o FPTAS para boa relação entre qualidade e tempo e o *branch-and-bound* para instâncias pequenas quando se deseja uma solução ótima para o problema.

6. Trabalhos Futuros

Como trabalhos futuros, seria ideal que se buscasse uma otimização das implementações dos algoritmos além de uma análise com diversas intâncias com carecterístcas mais diversas possíveis, além de testes com diferentes hadwares, linguagens de programação, compiladores e interpretadores.

Referências

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition.

Vimieiro, R. (2025a). Aula 12 – soluções exatas para problemas difíceis (branch-and-bound). [Online; accessed 04-julho-2025].

Vimieiro, R. (2025b). Aula 13 – soluções aproximadas para problemas difíceis. [Online; accessed 04-julho-2025].

Vimieiro, R. (2025c). Aula 14 – soluções aproximadas para problemas difíceis (parte 2). [Online; accessed 04-julho-2025].

Vimieiro, R. (2025d). Aula 15 – soluções aproximadas para problemas difíceis (parte 3). [Online; accessed 04-julho-2025].

Wikipédia (2025). Algoritmo de aproximação — wikipédia, a enciclopédia livre. [Online; accessed 06-julho-2025].