Algoritmos II - Trabalho Prático 2
Soluções para problemas difíceis

João Correia Costa[1], **Lucas Ferreira Pedras**[2], **Lucas Rafael Costa Santos**[3]

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

Abstract. *This work implements and compares three algorithms for the NP-hard Knapsack Problem: the exact Branch-and-Bound (B&B) solution, a 2-approximation algorithm, and a Fully Polynomial-Time Approximation Scheme (FPTAS). The empirical analysis validates the theoretical trade-offs, demonstrating that B&B is infeasible for large instances, while the FPTAS provides a superior balance between runtime and precision compared to the 2-approximation heuristic, serving as a guide for selecting the best approach per application.*

Resumo. *Este trabalho implementa e compara três algoritmos para o Problema da Mochila (NP-difícil): a solução exata (Branch-and-Bound), um algoritmo 2-aproximativo e um FPTAS. A análise empírica valida os trade-offs teóricos, mostrando que o B&B é inviável para instâncias grandes, enquanto o FPTAS oferece um equilíbrio superior entre tempo e precisão em comparação com a heurística 2-aproximativa, servindo como um guia para a escolha da melhor abordagem por aplicação.*

1. Introdução

O Problema da Mochila 0-1 (*0-1 Knapsack Problem*) é um dos problemas de otimização combinatória mais conhecidos e estudados na ciência da computação. Ele consiste que, dados n itens, cada um com um peso e valor associado, e uma mochila com capacidade W quilos, deseja-se maximizar o lucro total, respeitando a capacidade da mochila. Sua relevância ultrapassa o campo teórico, servindo como modelo para uma ampla gama de aplicações práticas, como alocação de recursos, seleção de projetos de investimento, carregamento de contêineres e otimização de orçamentos.

Embora simples em seu modelo, o Problema da Mochila 0-1 é classificado como NP-difícil, o que implica que não há algoritmos conhecidos que possam encontrar a solução ideal em tempo polinomial para todos os casos. Isto na prática implica que, para casos com um grande número de itens, um algoritmo exato poderia demorar demasiado tempo de execução. Essa dificuldade computacional motiva a exploração de diferentes paradigmas algorítmicos, estabelecendo um importante *trade-off* entre a otimalidade da solução e a eficiência computacional.

Este trabalho tem como objetivo implementar e avaliar empiricamente três abordagens distintas para resolver o Problema da Mochila. A primeira é um algoritmo exato baseado na técnica de *Branch-and-Bound*, que garante a obtenção da solução ótima por meio de uma busca inteligente no espaço de soluções. As outras duas são abordagens aproximativas: um algoritmo 2-aproximativo, que oferece uma solução eficiente

com garantia de qualidade constante, e um Esquema de Aproximação em Tempo Totalmente Polinomial (*Fully Polynomial Time Approximation Scheme* - FPTAS), que permite controlar o erro da solução em troca de maior tempo de processamento.

A comparação entre essas abordagens será conduzida com base em três métricas de desempenho, sendo elas tempo de execução, consumo de memória e qualidade da solução. Com isso, busca-se entender os limites práticos de cada estratégia e identificar em quais cenários cada uma delas apresenta melhor desempenho.

O restante deste artigo está organizado da seguinte forma. A Seção 2, que descreve as abordagens implementadas, detalhando as decisões de projeto envolvidas. A Seção 3, a qual apresenta a metodologia experimental adotada e discute os resultados obtidos e, por fim, a Seção 4, que por sua vez, reúne as conclusões e observações finais deste trabalho.

2. Abordagens Implementadas

Para solucionar o Problema da Mochila 0-1, foram implementadas três abordagens algorítmicas distintas, conforme requerido no enunciado do trabalho. Esta seção descreve a lógica e as principais decisões de projeto envolvidas nas implementações do algoritmo exato *Branch-and-Bound*, do algoritmo 2-aproximativo e do esquema de aproximação em tempo totalmente polinomial (FPTAS). Todas as soluções foram desenvolvidas em Python 3, utilizando as bibliotecas `NumPy` para operações numéricas e `heapq` para manipulação eficiente de estruturas de dados.

2.1. Branch-and-Bound

O algoritmo *Branch-and-Bound* (B&B) foi utilizado como abordagem exata, sendo capaz de encontrar a solução ótima para qualquer instância. O espaço de busca é representado por uma árvore binária, em que cada nível corresponde à decisão de incluir ou não um item específico na mochila. Para tornar a busca viável mesmo em instâncias maiores, foram adotadas estratégias de poda e priorização de nós promissores.

2.1.1. Estratégia de Busca

Optou-se por uma estratégia de busca do tipo *Best-First Search*. Essa abordagem expande prioritariamente o nó com maior potencial de levar à solução ótima, definido por uma estimativa superior (bound). Apesar de seu maior consumo de memória em relação à estratégia *Depth-First*, o *Best-First* tende a encontrar soluções ótimas mais rapidamente e permite podas mais eficazes nas subárvores menos promissoras.

2.1.2. Estrutura de Dados

Para o controle da fronteira de busca, foi utilizada uma fila de prioridade (*priority queue*) implementada com a biblioteca `heapq`, que fornece uma *min-heap*. Como o problema requer uma *max-heap* (nós com maior bound devem ser explorados primeiro), foi adotada a técnica de armazenar os valores de bound com sinal negativo. Essa abordagem permite simular o comportamento desejado de forma eficiente e compatível com a biblioteca padrão.

2.1.3. Função de Limite (Upper Bound)

A função de bound utilizada baseia-se na relaxação do problema da mochila fracionária. Antes da execução, os itens são ordenados de forma decrescente segundo sua razão valor/peso. O bound de um nó é calculado somando-se ao valor atual o valor obtido ao preencher a capacidade restante da mochila com frações dos itens subsequentes, seguindo a ordem de prioridade. Essa heurística fornece uma estimativa admissível e eficiente do valor máximo possível a partir daquele ponto da árvore.

2.2. Esquema de Aproximação em Tempo Totalmente Polinomial (FPTAS)

Para permitir o controle da qualidade da solução, foi implementado um esquema de aproximação FPTAS (*Fully Polynomial Time Approximation Scheme*). Essa abordagem permite configurar o erro relativo admissível $\varepsilon > 0$, garantindo que o valor da solução gerada seja, no mínimo, $(1 - \varepsilon)$ vezes o valor da solução ótima.

A implementação segue dois passos principais:

1. **Escalonamento dos Valores:** O valor máximo entre os itens (v_{max}) é utilizado para definir um fator de escala μ, calculado com base em ε e no número de itens n. Os valores originais v_i são transformados em novos valores discretizados $v'_i = \lfloor v_i/\mu \rfloor$, o que reduz o espaço de estados da programação dinâmica subsequente.
2. **Programação Dinâmica:** Com os valores discretizados v', aplica-se uma abordagem clássica de programação dinâmica, com complexidade $O(n \cdot V'_{\text{max}})$, onde V'_{max} é o maior valor escalonado. O algoritmo calcula o peso mínimo necessário para atingir cada valor possível, e em seguida seleciona o valor máximo que respeita a capacidade da mochila. A solução é então mapeada de volta para os índices originais dos itens, fornecendo uma aproximação eficiente para o problema original.

Vantagens e Complexidade

A principal vantagem do FPTAS está em seu controle explícito sobre a qualidade da solução, por meio do parâmetro ε. Isso permite ajustar o algoritmo conforme o nível de precisão desejado: quanto menor o erro permitido, maior a fidelidade à solução ótima. Além disso, por ser um esquema de aproximação com garantia teórica, o FPTAS oferece uma alternativa viável ao uso de algoritmos exatos em instâncias grandes, mantendo um erro controlado.

No entanto, esse benefício vem acompanhado de um custo computacional maior. A complexidade do FPTAS é $O(n^2/\varepsilon)$, assumindo que os valores dos itens sejam inteiros e que o fator de escala μ seja proporcional a $\varepsilon \cdot v_{\text{max}}/n$. Como o escalonamento reduz o espaço de estados da programação dinâmica, essa complexidade torna-se aceitável para ε moderados. Ainda assim, o desempenho pode se degradar para valores muito pequenos de ε, tornando o FPTAS inadequado em situações com restrições severas de tempo de execução.

2.3. Algoritmo 2-Aproximativo

Conforme solicitado, também foi implementado um algoritmo com fator de aproximação garantido igual a 2. A estratégia adotada consiste em combinar duas soluções heurísticas

distintas, selecionando ao final aquela que apresentar maior valor total:

1. **Heurística Gulosa por Razão Valor/Peso:** Os itens são ordenados de forma decrescente segundo sua razão valor/peso, e são adicionados à mochila enquanto houver capacidade disponível. Essa abordagem tem complexidade $O(n \log n)$, devido à ordenação inicial.
2. **Melhor Item Individual:** Em paralelo, identifica-se o item de maior valor que pode ser incluído isoladamente na mochila, ou seja, cujo peso não ultrapasse a capacidade.

A solução final é a de maior valor entre as duas alternativas descritas. Essa técnica garante que o valor da solução seja, no mínimo, 50% do valor ótimo, caracterizando um algoritmo 2-aproximativo com baixa complexidade e desempenho robusto em instâncias grandes.

Vantagens e Complexidade

A principal vantagem do algoritmo 2-aproximativo é sua simplicidade e velocidade de execução. Por evitar qualquer tipo de busca combinatória ou programação dinâmica, ele é extremamente eficiente, sendo apropriado para instâncias de grande escala ou aplicações em tempo real. Além disso, sua implementação é direta, utilizando apenas ordenação e operações básicas.

Sua complexidade é dominada pela etapa de ordenação dos itens por razão valor/peso, resultando em tempo $O(n \log n)$. O restante do algoritmo (seleção gulosa e busca pelo melhor item individual) opera em tempo linear $O(n)$.

Apesar de não oferecer controle sobre o erro relativo, o algoritmo possui garantia teórica de que a solução obtida será, no mínimo, 50% da ótima. Essa limitação o torna menos atrativo quando a qualidade da solução é prioridade, mas seu uso é justificável quando a velocidade é um fator crítico.

3. Experimentos e Análise de Resultados

Esta seção apresenta os resultados obtidos com as três abordagens implementadas para o Problema da Mochila 0-1, com foco em três métricas principais: tempo de execução, consumo de memória e qualidade da solução. Os testes foram conduzidos sobre 13 instâncias de tamanhos variados (de $n = 4$ a $n = 500$), incluindo instâncias *low-dimensional* e *large-scale*, com tempo limite de 30 minutos por execução e medição do pico de memória via `tracemalloc`.

3.1. Tempo de Execução

A Figura 1 mostra o tempo de execução do algoritmo *Branch-and-Bound* em escala logarítmica. Nota-se que, apesar do bom desempenho para instâncias pequenas, o tempo cresce rapidamente com o número de itens, chegando a quase 100 segundos em alguns casos com $n = 50$.

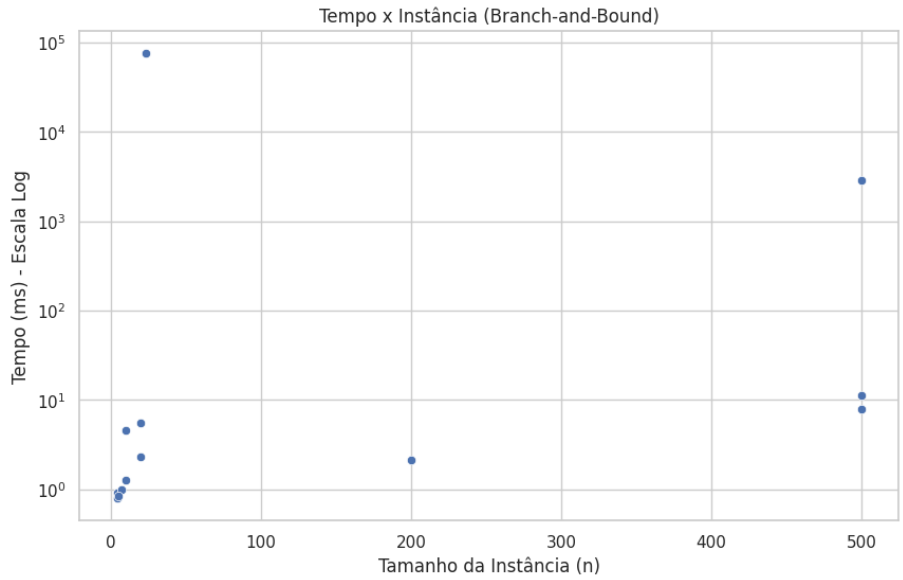

Figure 1. Tempo de execução do Branch-and-Bound em função do tamanho da instância.

A Figura 2 exibe os resultados para o algoritmo 2-aproximativo, que apresentou desempenho excelente em todas as instâncias, com tempo sempre inferior a 4 ms. A estabilidade observada demonstra sua viabilidade para aplicações em tempo real.

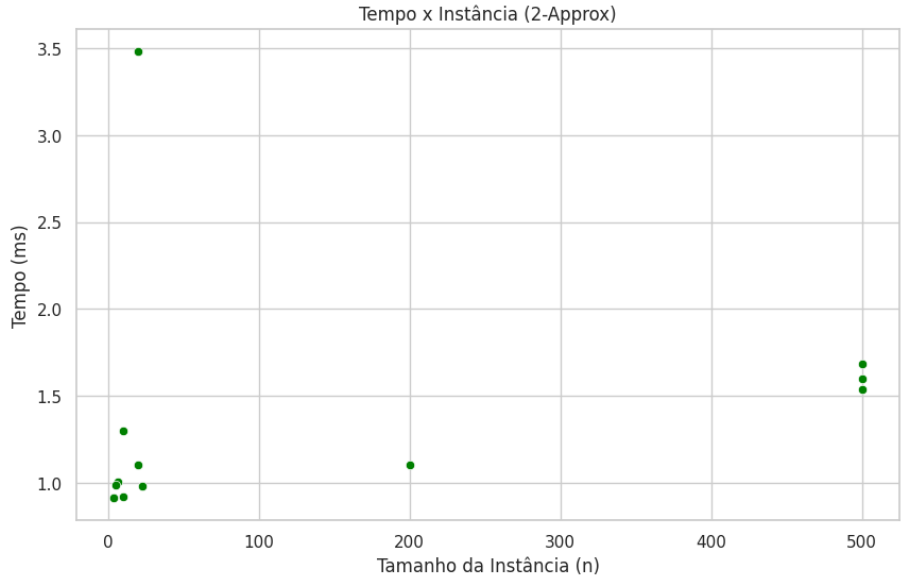

Figure 2. Tempo de execução do algoritmo 2-aproximativo.

A Figura 3 compara o FPTAS para diferentes valores de ε. Observa-se um crescimento acentuado do tempo com a redução do erro permitido. Para $\varepsilon = 0{,}1$, o tempo chegou a ultrapassar 22 minutos em instâncias com $n = 500$, tornando essa configuração pouco prática.

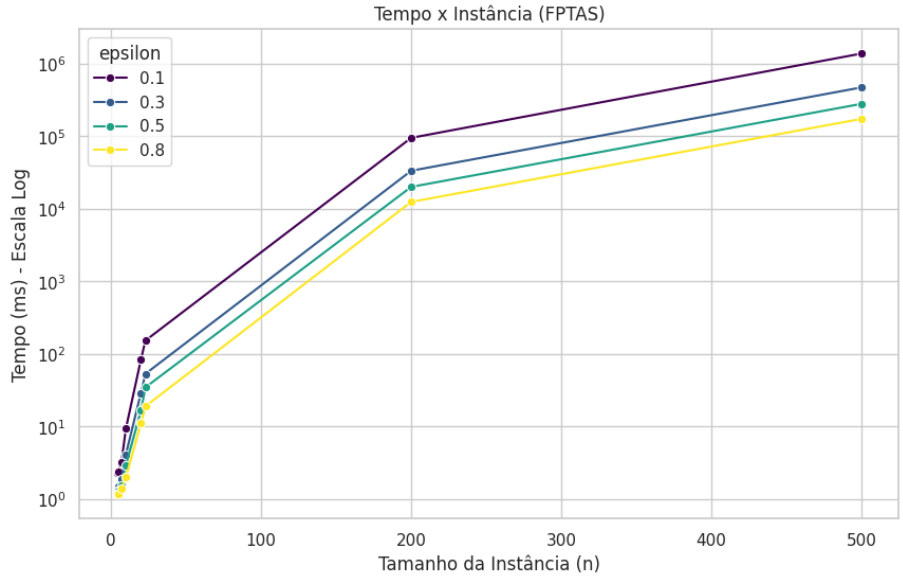

Figure 3. Tempo de execução do FPTAS para diferentes valores de ε.

3.2. Consumo de Memória

A Figura 4 apresenta o pico de memória (em KB, escala log) para todos os algoritmos. Como esperado, o FPTAS teve os maiores consumos, especialmente para $\varepsilon = 0{,}1$, com picos superiores a 3 GB

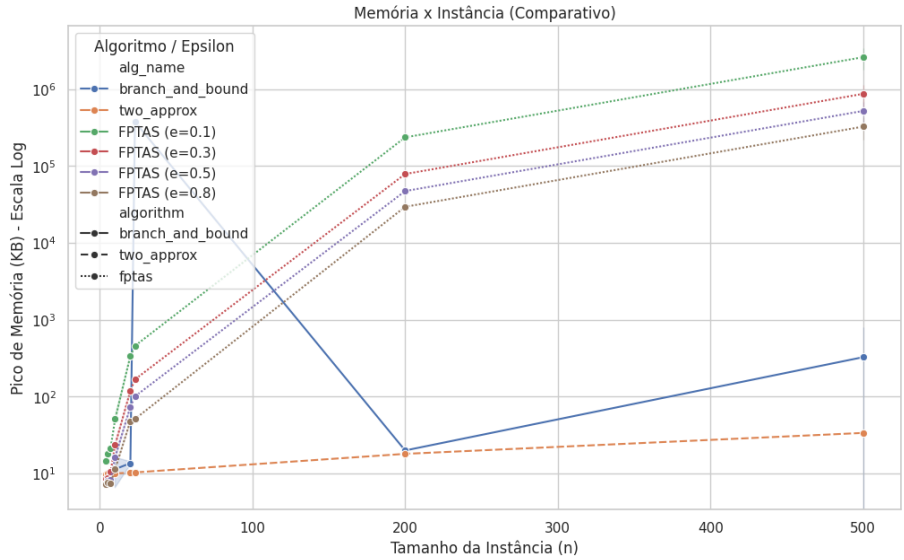

Figure 4. Consumo máximo de memória por algoritmo em função do tamanho da instância.

3.3. Qualidade da Solução

A Figura 5 mostra o fator de aproximação para cada algoritmo, definido como a razão entre o valor ótimo e o valor encontrado. O FPTAS demonstrou altíssima precisão, com o fator de aproximação médio mantendo-se em $1{,}00$ para valores de ε até $0{,}5$, e subindo

para apenas $1{,}01$ com $\varepsilon = 0{,}8$, validando empiricamente sua robustez. O algoritmo 2-aproximativo, apesar de eficiente, apresentou variações mais significativas em instâncias pequenas, chegando a fator de $1{,}47$ (erro de 47%).

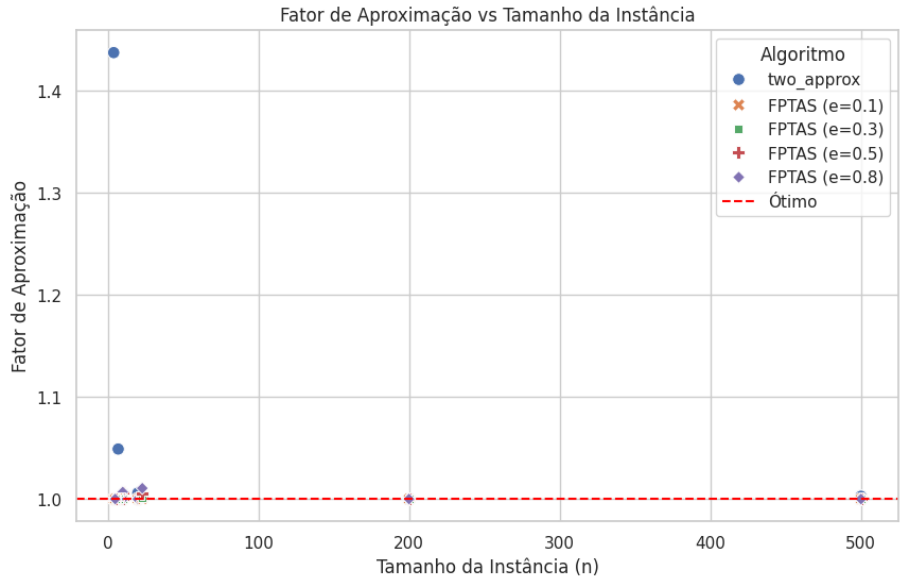

Figure 5. Fator de aproximação (ótimo/obtido) para os algoritmos aproximativos.

3.4. Resumo Comparativo

A Tabela 1 resume os principais indicadores estatísticos coletados ao longo dos experimentos, enquanto a Tabela 2 apresenta todos os resultados obtidos.

Table 1. Resumo estatístico: tempo, memória e fator de aproximação.

Algoritmo	Tempo Médio (ms)	Memória (KB)	Fator Médio	Erro Médio (%)	Erro Máx. (%)
Branch-and-Bound	5.997	29.038	1.00	0.00	0.00
FPTAS (ε=0.1)	326560.04	619610.58	1.00	0.00	0.03
FPTAS (ε=0.3)	111520.90	206551.95	1.00	0.06	0.68
FPTAS (ε=0.5)	66189.93	123867.02	1.00	0.09	0.68
FPTAS (ε=0.8)	41067.39	77374.86	1.00	0.13	1.04
2-Aproximativo	1.35	16.11	1.04	2.88	30.43

3.5. Discussão

Com base nos resultados, é possível delinear um perfil claro de uso para cada abordagem. O Branch-and-Bound, por exemplo, se mostrou altamente preciso, mas sua escalabilidade é limitada, tornando-o ideal apenas para instâncias pequenas onde a otimalidade é mandatória. Em contrapartida, o algoritmo 2-aproximativo se destaca pela velocidade em qualquer cenário, sendo a melhor escolha quando o tempo de resposta é crítico e a

aplicação suporta maiores margens de erro. Finalmente, o FPTAS se posiciona como a solução mais robusta e flexível, ideal para instâncias grandes onde se pode 'pagar' com mais tempo computacional para obter uma garantia de qualidade configurável e próxima da ótima.

4. Conclusão

Este trabalho apresentou a implementação e análise comparativa de três algoritmos para o Problema da Mochila 0-1: uma abordagem exata com *Branch-and-Bound*, um algoritmo 2-aproximativo e um esquema de aproximação em tempo totalmente polinomial (FPTAS). A avaliação empírica contemplou diferentes instâncias de tamanhos variados, considerando tempo de execução, consumo de memória e qualidade da solução.

Os resultados confirmaram o perfil teórico de cada algoritmo. O *Branch-and-Bound* se mostrou altamente eficaz para instâncias pequenas, produzindo soluções ótimas com baixo consumo de memória, porém, seu tempo de execução cresceu exponencialmente em instâncias maiores, tornando-o inviável para aplicações em larga escala.

O algoritmo 2-aproximativo destacou-se pela sua extrema eficiência computacional, de modo que, mesmo em instâncias grandes, seu tempo de execução permaneceu inferior a 2 ms, com consumo de memória quase constante. Por outro lado, sua qualidade de solução variou mais, apresentando desvios de até 30% em relação ao valor ótimo em alguns casos.

O FPTAS demonstrou-se uma alternativa equilibrada entre precisão e custo computacional. Para valores baixos de ε, a aproximação foi praticamente exata, porém com um tempo de execução elevado, enquanto para valores maiores de ε, o tempo e a memória foram reduzidos consideravelmente, mantendo ainda uma qualidade de solução satisfatória.

Em síntese, a escolha do algoritmo ideal depende dos requisitos da aplicação: o *Branch-and-Bound* é indicado quando a otimalidade é imprescindível e o tempo não é um fator crítico; o 2-aproximativo é apropriado em contextos que demandam respostas rápidas; já o FPTAS é ideal quando se deseja controle sobre o erro e se dispõe de mais tempo de processamento.

Como trabalho futuro, sugere-se explorar versões paralelas dos algoritmos apresentados, avaliar instâncias com restrições adicionais e investigar outras heurísticas e esquemas de aproximação para o problema da mochila.

References

[1] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Introduction to Algorithms*. 3rd ed. MIT Press, 2009.

[2] KLEINBERG, J.; TARDOS, É. *Algorithm Design*. 1st ed. Addison-Wesley, 2005.

[3] LEVITIN, A. *Introduction to the Design and Analysis of Algorithms*. 3rd ed. Pearson, 2012.

[4] ORTEGA, J. *Instances for 0/1 Knapsack Problem (Low-dimensional)*. Disponível em: `http://artemisa.unicauca.edu.co/~johnyortega/instances_01_KP/`. Acesso em: 04 jul. 2025.

[5] SCOVINO, L. *Large-scale 0/1 Knapsack Problems*. Kaggle. Disponível em: `https://www.kaggle.com/datasets/sc0v1n0/large-scale-01-knapsack-problems`. Acesso em: 04 jul. 2025.

[6] VIMIEIRO, R. *Material da disciplina DCC207 - Algoritmos 2*. Universidade Federal de Minas Gerais, 2025.

[7] ARBEX, M. *Dicas para a Escrita de Artigos Científicos*. Disponível em: `https://homepages.dcc.ufmg.br/~mirella/doku.php?id=escrita`. Acesso em: 04 jul. 2025.

Table 2: Resultados detalhados por instância, algoritmo e configuração.

Instance	n	Algoritmo	ε	Valor	Tempo (ms)	Memória (KB)
f10_l-d_kp_20_879	20	branch_and_bound	–	1025	5.53	14.07
f10_l-d_kp_20_879	20	two_approx	–	1019	3.48	10.25
f10_l-d_kp_20_879	20	fptas	0.1	1025	83.38	339.86
f10_l-d_kp_20_879	20	fptas	0.3	1025	28.36	119.40
f10_l-d_kp_20_879	20	fptas	0.5	1025	16.04	74.19
f10_l-d_kp_20_879	20	fptas	0.8	1025	10.96	48.56
f1_l-d_kp_10_269	10	branch_and_bound	–	295	1.26	7.96
f1_l-d_kp_10_269	10	two_approx	–	294	0.92	10.01
f1_l-d_kp_10_269	10	fptas	0.1	295	9.67	54.38
f1_l-d_kp_10_269	10	fptas	0.3	293	4.08	22.56
f1_l-d_kp_10_269	10	fptas	0.5	293	2.57	15.33
f1_l-d_kp_10_269	10	fptas	0.8	293	1.93	11.58
f2_l-d_kp_20_878	20	branch_and_bound	–	1024	2.34	13.16
f2_l-d_kp_20_878	20	two_approx	–	1018	1.10	10.25
f2_l-d_kp_20_878	20	fptas	0.1	1024	83.09	338.87
f2_l-d_kp_20_878	20	fptas	0.3	1024	29.43	118.72
f2_l-d_kp_20_878	20	fptas	0.5	1024	16.86	73.34
f2_l-d_kp_20_878	20	fptas	0.8	1024	10.97	47.58
f3_l-d_kp_4_20	4	branch_and_bound	–	35	0.79	7.62
f3_l-d_kp_4_20	4	two_approx	–	35	0.91	9.86
f3_l-d_kp_4_20	4	fptas	0.1	35	2.57	14.79
f3_l-d_kp_4_20	4	fptas	0.3	35	1.35	8.76
f3_l-d_kp_4_20	4	fptas	0.5	35	1.16	7.36
f3_l-d_kp_4_20	4	fptas	0.8	35	1.08	7.15
f4_l-d_kp_4_11	4	branch_and_bound	–	23	0.91	7.62
f4_l-d_kp_4_11	4	two_approx	–	16	0.91	9.86
f4_l-d_kp_4_11	4	fptas	0.1	23	2.05	14.24

Instance	n	Algoritmo	ε	Valor	Tempo (ms)	Memória (KB)
f4_l-d_kp_4_11	4	fptas	0.3	23	1.24	8.40
f4_l-d_kp_4_11	4	fptas	0.5	23	1.44	7.40
f4_l-d_kp_4_11	4	fptas	0.8	23	1.23	7.15
f6_l-d_kp_10_60	10	branch_and_bound	–	52	4.56	14.79
f6_l-d_kp_10_60	10	two_approx	–	52	1.30	10.01
f6_l-d_kp_10_60	10	fptas	0.1	52	9.01	47.42
f6_l-d_kp_10_60	10	fptas	0.3	52	4.16	24.39
f6_l-d_kp_10_60	10	fptas	0.5	52	3.28	17.25
f6_l-d_kp_10_60	10	fptas	0.8	52	2.05	11.15
f7_l-d_kp_7_50	7	branch_and_bound	–	107	1.00	7.67
f7_l-d_kp_7_50	7	two_approx	–	102	1.00	9.93
f7_l-d_kp_7_50	7	fptas	0.1	107	3.21	21.15
f7_l-d_kp_7_50	7	fptas	0.3	107	1.86	10.43
f7_l-d_kp_7_50	7	fptas	0.5	107	1.55	8.28
f7_l-d_kp_7_50	7	fptas	0.8	107	1.37	7.47
f8_l-d_kp_23_10000	23	branch_and_bound	–	9767	75079.70	376412.07
f8_l-d_kp_23_10000	23	two_approx	–	9751	0.98	10.32
f8_l-d_kp_23_10000	23	fptas	0.1	9764	152.36	457.38
f8_l-d_kp_23_10000	23	fptas	0.3	9756	52.96	167.52
f8_l-d_kp_23_10000	23	fptas	0.5	9714	34.45	100.70
f8_l-d_kp_23_10000	23	fptas	0.8	9665	18.85	51.13
f9_l-d_kp_5_80	5	branch_and_bound	–	130	0.85	7.64
f9_l-d_kp_5_80	5	two_approx	–	130	0.99	9.88
f9_l-d_kp_5_80	5	fptas	0.1	130	2.39	18.04
f9_l-d_kp_5_80	5	fptas	0.3	130	1.46	8.78
f9_l-d_kp_5_80	5	fptas	0.5	130	1.28	7.96
f9_l-d_kp_5_80	5	fptas	0.8	130	1.18	7.55
knapPI_14_200_1000_1_info.csv	200	branch_and_bound	–	5397	2.15	19.92
knapPI_14_200_1000_1_info.csv	200	two_approx	–	5397	1.10	18.00
knapPI_14_200_1000_1_info.csv	200	fptas	0.1	5397	95010.58	236041.79
knapPI_14_200_1000_1_info.csv	200	fptas	0.3	5397	33430.20	78690.59
knapPI_14_200_1000_1_info.csv	200	fptas	0.5	5397	20027.59	47214.43
knapPI_14_200_1000_1_info.csv	200	fptas	0.8	5397	12438.05	29502.58
knapPI_1_500_1000_1_info.csv	500	branch_and_bound	–	28857	11.14	53.54
knapPI_1_500_1000_1_info.csv	500	two_approx	–	28834	1.68	33.69
knapPI_1_500_1000_1_info.csv	500	fptas	0.1	28857	1321785.41	1932707.60
knapPI_1_500_1000_1_info.csv	500	fptas	0.3	28857	434240.36	644902.96
knapPI_1_500_1000_1_info.csv	500	fptas	0.5	28857	254985.05	386666.39
knapPI_1_500_1000_1_info.csv	500	fptas	0.8	28857	159306.87	241377.98
knapPI_2_500_1000_1_info.csv	500	branch_and_bound	–	4566	7.95	50.09
knapPI_2_500_1000_1_info.csv	500	two_approx	–	4552	1.60	33.69
knapPI_2_500_1000_1_info.csv	500	fptas	0.1	4566	1348717.12	2344261.42
knapPI_2_500_1000_1_info.csv	500	fptas	0.3	4566	448278.25	780377.11
knapPI_2_500_1000_1_info.csv	500	fptas	0.5	4566	261322.67	467661.34
knapPI_2_500_1000_1_info.csv	500	fptas	0.8	4566	157847.68	292161.74
knapPI_3_500_1000_1_info.csv	500	branch_and_bound	–	7117	2837.96	873.59
knapPI_3_500_1000_1_info.csv	500	two_approx	–	7098	1.54	33.69
knapPI_3_500_1000_1_info.csv	500	fptas	0.1	7117	1479419.68	3540620.62
knapPI_3_500_1000_1_info.csv	500	fptas	0.3	7117	533698.04	1180715.78
knapPI_3_500_1000_1_info.csv	500	fptas	0.5	7117	324055.09	708417.33
knapPI_3_500_1000_1_info.csv	500	fptas	0.8	7117	204233.87	442631.50