Análise de Algoritmos para o Problema da Mochila Binária

Eduardo Assis Tomich, Raphael Alves dos Reis, Victor Yuji Yano

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

{edtomich, cap497, victory}@ufmg.br

Abstract. *This work presents a comprehensive study of algorithms for solving the 0-1 Knapsack Problem, covering both exact and approximate strategies. We implemented and analyzed three main approaches: an exact Branch-and-Bound algorithm, a lightweight 2-approximation heuristic, and a Fully Polynomial Time Approximation Scheme (FPTAS) configured with different values of* ϵ*. The experiments involved instances with varying numbers of items, evaluating each algorithm's performance in terms of execution time, memory usage, and solution quality measured via approximation factor relative to the known optimum. The comparative analysis aims to highlight practical trade-offs between solution precision and computational costs, providing guidance for selecting appropriate strategies in different scenarios.*

Resumo. *Este trabalho apresenta um estudo abrangente sobre algoritmos para resolver o Problema da Mochila Binária (0-1 Knapsack Problem), abordando estratégias exatas e aproximativas. Foram implementadas e analisadas três abordagens principais: um algoritmo exato baseado em Branch-and-Bound, um heurístico 2-aproximativo com baixo custo computacional e um FPTAS (Fully Polynomial Time Approximation Scheme) configurado para diferentes valores de* ϵ*. Os experimentos incluíram instâncias com diferentes números de itens, avaliando o desempenho de cada algoritmo em termos de tempo de execução, uso de memória e qualidade da solução medida via fator de aproximação em relação ao ótimo conhecido. A análise comparativa busca evidenciar os trade-offs práticos entre precisão da solução e custos computacionais, oferecendo subsídios para a escolha informada de estratégias em diferentes cenários.*

1. Introdução

O Problema da Mochila Inteira (0-1 Knapsack Problem) é um dos problemas clássicos de otimização combinatória e programação inteira, sendo amplamente estudado em ciência da computação, pesquisa operacional e áreas afins. Em sua formulação tradicional, o problema consiste em selecionar um subconjunto de itens, cada um com um valor associado e um peso, de forma a maximizar o valor total transportado sem exceder a capacidade máxima da mochila.

Formalmente, pode ser descrito pela seguinte formulação matemática:

$$\max \sum_{i=1}^{n} v_i x_i \quad \text{sujeito a} \quad \sum_{i=1}^{n} w_i x_i \leq C, \quad x_i \in \{0, 1\},$$

onde:

- n representa o número de itens disponíveis;
- v_i é o valor do item i;
- w_i é o peso do item i;
- C é a capacidade máxima da mochila;
- x_i indica se o item i está incluído (1) ou não (0) na solução.

Apesar de sua formulação simples, o Problema da Mochila é conhecido por sua dificuldade computacional, sendo classificado como NP-completo. Seu espaço de busca possui 2^n soluções possíveis, o que evidencia a explosão combinatória envolvida: para instâncias grandes ou com características adversariais, métodos exatos podem se tornar inviáveis, exigindo tempo exponencial em relação ao número de itens. Por isso, há grande interesse em estratégias heurísticas ou aproximadas, que ofereçam garantias de qualidade em tempo polinomial.

Este trabalho foi desenvolvido com o objetivo de estudar, implementar e comparar diferentes algoritmos para resolver o Problema da Mochila Binária, com foco especial em explorar o compromisso entre precisão da solução e custos computacionais (tempo e memória). Foram consideradas tanto abordagens exatas quanto heurísticas e esquemas de aproximação formal, buscando evidenciar as vantagens e limitações de cada uma delas.

Especificamente, o estudo concentrou-se em três tipos de algoritmos:

- **Branch-and-Bound**: Um método exato que explora sistematicamente o espaço de soluções através de uma árvore de decisão, aplicando técnicas de poda para eliminar subárvores que não podem conter soluções melhores que a atual. Apesar de garantir a optimalidade, seu tempo de execução pode crescer exponencialmente, tornando-o inviável em instâncias grandes.
- **Novo 2-Aproximativo**: Uma heurística simples e eficiente, desenvolvida pelos autores, baseada em uma estratégia gulosa que prioriza itens com melhor relação valor/peso. Essa abordagem garante teoricamente um fator de aproximação de no máximo 2, sendo muito rápida e com baixo consumo de memória.
- **FPTAS (Fully Polynomial Time Approximation Scheme)**: Um esquema aproximativo que permite ao usuário especificar um parâmetro $\epsilon > 0$, determinando o grau de aproximação desejado. O FPTAS oferece garantia formal de que a solução obtida será no mínimo $(1 - \epsilon)$ vezes o valor ótimo, com complexidade polinomial em n e $1/\epsilon$.

Para conduzir uma análise abrangente, foram utilizados dois conjuntos de teste:

- **Instâncias pequenas**: com menos de 25 itens, permitindo uso eficiente de métodos exatos e comparação direta com soluções ótimas conhecidas.
- **Instâncias grandes**: com até 500 itens, mais representativas de aplicações práticas, desafiando algoritmos exatos e testando a escalabilidade dos métodos aproximativos.

Essas instâncias foram obtidas de conjuntos disponíveis publicamente em `http://artemisa.unicauca.edu.co/~johnyortega/instances_01_KP/` (instâncias pequenas) e `https://www.kaggle.com/datasets/sc0v1n0/large-scale-01-knapsack-problems` (instâncias grandes).

Esses dois grupos possibilitaram avaliar como o desempenho dos algoritmos varia de acordo com o tamanho da instância, destacando limitações práticas e trade-offs inerentes às abordagens escolhidas.

A motivação para este trabalho também inclui aspectos didáticos e práticos. Do ponto de vista acadêmico, o Problema da Mochila é um excelente exemplo para ilustrar conceitos de complexidade computacional, técnicas de poda, estratégias gulosas e aproximação via programação dinâmica. Já em termos aplicados, problemas de mochila aparecem em contextos como logística, alocação de recursos, orçamentação, planejamento de produção, entre outros.

Os experimentos foram planejados para cobrir diferentes aspectos do problema:

- Avaliar a qualidade da solução obtida (via fator de aproximação em relação ao ótimo conhecido).
- Medir o tempo de execução, considerando limites práticos (inclusive com timeout de 30 minutos para métodos exatos).
- Analisar o consumo de memória, especialmente em métodos que utilizam tabelas de programação dinâmica ou estruturas auxiliares complexas.

Outro objetivo central foi investigar como características específicas das instâncias influenciam o comportamento dos algoritmos. Por exemplo:

- Relação entre número de itens e explosão combinatória do Branch-and-Bound.
- Impacto de valores e pesos homogêneos ou heterogêneos na eficiência de estratégias gulosas.
- Sensibilidade do FPTAS ao parâmetro ϵ e como ele afeta tempo, memória e qualidade de solução.

Por fim, este relatório documenta não apenas os resultados experimentais obtidos, mas também o processo de desenvolvimento dos algoritmos, suas estratégias de implementação, escolhas metodológicas (como ordenação ou não dos itens por razão valor/peso) e ferramentas utilizadas para medição de desempenho (tempo e memória).

Com isso, busca-se oferecer um estudo abrangente que permita ao leitor compreender tanto os fundamentos teóricos das abordagens quanto os resultados práticos, servindo como referência para decisões informadas sobre qual algoritmo utilizar em diferentes cenários.

2. Descrição das Abordagens

Nesta seção detalhamos as abordagens escolhidas para resolver o Problema da Mochila Binária, apresentando ideia central, estratégia, complexidade e garantias. Ao final, uma tabela resume suas principais características.

Visão geral:

- **Backtracking**: método exato por enumeração completa.
- **Branch-and-Bound**: método exato com poda inteligente.
- **Novo 2-Aproximativo**: heurística gulosa com fator de aproximação ≤ 2.
- **FPTAS**: esquema aproximativo ajustável via ϵ com garantias formais.

2.1. Backtracking

Busca exaustiva em árvore binária de profundidade n.

- **Ideia**: explora todas as combinações de inclusão/exclusão.
- **Complexidade**: tempo exponencial $O(2^n)$, espaço $O(n)$.
- **Vantagem**: encontra sempre o ótimo.
- **Limite**: inviável para instâncias grandes (usa timeout).

Algorithm 1 Backtracking para Mochila Binária

Require: v, w, C
Ensure: Melhor valor e solução
1: $melhor\_valor \leftarrow 0$
2: Definir $Busca(indice, valor\_atual, peso\_atual, solucao\_atual)$
3: **if** peso_atual $> C$ **then**
4: Retornar
5: **end if**
6: **if** valor_atual $\geq$ melhor_valor **then**
7: Atualizar
8: **end if**
9: **if** indice $< n$ **then**
10: $Busca$ com inclusão
11: $Busca$ sem inclusão
12: **end if**

2.2. Branch-and-Bound

Método exato com poda para reduzir explosão combinatória.

Ideia central:

- **Branch**: divide em subproblemas (inclusão/exclusão).
- **Bound**: estima limite superior (relaxação fracionária) para cortar ramos.

Estimativa de Bound:

$$\text{bound} = valor\_atual + \sum_{j=i}^{n} \min(w_j, capacidade\_restante) \times \frac{v_j}{w_j}.$$

Estratégia: DFS com ordenação por v_i/w_i para resultados mais estáveis.

Complexidade: tempo exponencial no pior caso; espaço $O(n)$.

Vantagem: encontra o ótimo com menos nós visitados que backtracking puro.

Algorithm 2 Branch-and-Bound para Mochila Binária

Require: v, w, C
Ensure: Melhor valor e solução
1: $melhor\_valor \leftarrow 0$
2: Definir $Busca(indice, valor\_atual, peso\_atual, solucao\_atual)$
3: **if** peso_atual $> C$ **then**
4: Retornar
5: **end if**
6: **if** valor_atual $\geq$ melhor_valor **then**
7: Atualizar
8: **end if**
9: Calcular $bound$
10: **if** bound $\leq$ melhor_valor **then**
11: Retornar
12: **end if**
13: **if** indice $< n$ **then**
14: $Busca$ com inclusão
15: $Busca$ sem inclusão
16: **end if**

Algorithm 3 Novo 2-Aproximativo para Mochila Binária

Require: v, w, C
Ensure: Valor aproximado e solução
1: Ordenar por v_i/w_i
2: Inicializar capacidade restante e vetor de inclusão
3: **for** item na lista ordenada **do**
4: **if** peso $\leq$ capacidade restante **then**
5: Incluir item
6: **end if**
7: **end for**
8: Retornar valor total e vetor

2.3. Novo 2-Aproximativo

Heurística gulosa extremamente simples e eficiente.

Ideia:

- Ordenar itens por v_i/w_i decrescente.
- Selecionar enquanto houver capacidade.

Justificativa do fator 2: Para qualquer solução ótima, ou existe um item muito valioso sozinho ou um conjunto de itens pequenos com boa razão valor/peso. A heurística simula isso implicitamente, garantindo valor $\geq$ OPT$/2$.

Complexidade: $O(n \log n)$ para ordenação + $O(n)$ para seleção; espaço $O(n)$.

2.4. FPTAS

Esquema aproximativo com controle explícito via ϵ.

Ideia:

- Escalonar valores para reduzir a dimensão do DP.
- Resolver problema escalado via DP para obter solução aproximada.

Garantia: Para qualquer $\epsilon > 0$, encontra solução com valor $\geq (1 - \epsilon) \times$ ótimo.

Escalonamento:

$$\text{scale} = \frac{\epsilon \cdot v_{\max}}{n}, \quad v_i' = \left\lfloor \frac{v_i}{\text{scale}} \right\rfloor$$

Complexidade: tempo e espaço $O(n^2/\epsilon)$.

Algorithm 4 FPTAS para Mochila Binária

Require: v, w, C, ϵ
Ensure: Valor aproximado e solução
1: Calcular $v_{\max}$ e $scale$
2: Escalar v_i'
3: Inicializar DP[0]=0
4: **for** cada item i **do**
5: **for** valor total possível (decrescente) **do**
6: **if** nova combinação respeita C e melhora peso **then**
7: Atualizar DP
8: **end if**
9: **end for**
10: **end for**
11: Reconstruir solução
12: Retornar valor e itens escolhidos

Algoritmo	Exato	Complexidade	Memória	Aproximação
Backtracking	Sim	Exponencial	Baixa	Ótimo
Branch-and-Bound	Sim	Exponencial	Moderada	Ótimo
Novo 2-Aproximativo	Não	$O(n \log n)$	Baixa	$\leq 2\times$ ótimo
FPTAS	Não	$O(n^2/\epsilon)$	Alta	$(1 - \epsilon)\times$ ótimo

Table 1. Comparação geral das abordagens implementadas.

3. Experimentos e Discussão dos Resultados

Nesta seção apresentamos a metodologia utilizada, os conjuntos de instâncias testados, os parâmetros de controle (como o tempo limite) e os resultados obtidos em termos de qualidade da solução (via fator de aproximação), tempo de execução e uso de memória. Em seguida, discutimos criticamente os limites observados em cada abordagem, comparando os algoritmos e avaliando o impacto do número de itens em seu desempenho.

3.1. Metodologia Experimental

Os experimentos foram implementados em Python 3.10, usando $psutil$ para medir memória (RSS), $time$ para cronometrar execuções e $multiprocessing$ para impor limites de tempo. Cada instância foi processada de forma independente para garantir medidas consistentes e isoladas.

Execução:

- Novo 2-Aproximativo
- FPTAS com $\epsilon \in (4.00, 2.00, 1.00, 0.50, 0.25)$
- Branch-and-Bound
- Backtracking

Limitação de tempo: Foi imposto um limite estrito de 1800 segundos (30 minutos) por instância para evitar execuções indefinidas em casos de explosão combinatória. O Backtracking puro atingiu esse limite em todas as instâncias grandes, resultando em ausência de solução para essas execuções.

3.2. Conjuntos de Instâncias

Foram utilizados dois grupos principais de instâncias: um conjunto de **instâncias pequenas** (até 23 itens) e outro de **instâncias grandes** (até 500 itens). Todas foram obtidas das fontes citadas na introdução. Para os gráficos e análises desta seção, unificamos os conjuntos em um só.

3.3. Métricas Coletadas

Para cada combinação (instância, algoritmo), registramos:

- Valor da solução encontrada.
- Fator de aproximação (razão entre valor ótimo conhecido e valor obtido).
- Tempo de execução (em segundos).
- Memória máxima usada (em megabytes).

3.4. Resultados Consolidados

Cada algoritmo, incluindo cada configuração de ϵ no FPTAS, é representado como uma linha ou conjunto de pontos distintos (com cor específica), permitindo comparações diretas quando resultados são obtidos. Em casos de timeout (como ocorreu no Backtracking para instâncias grandes), não há resultado incluído nos gráficos.

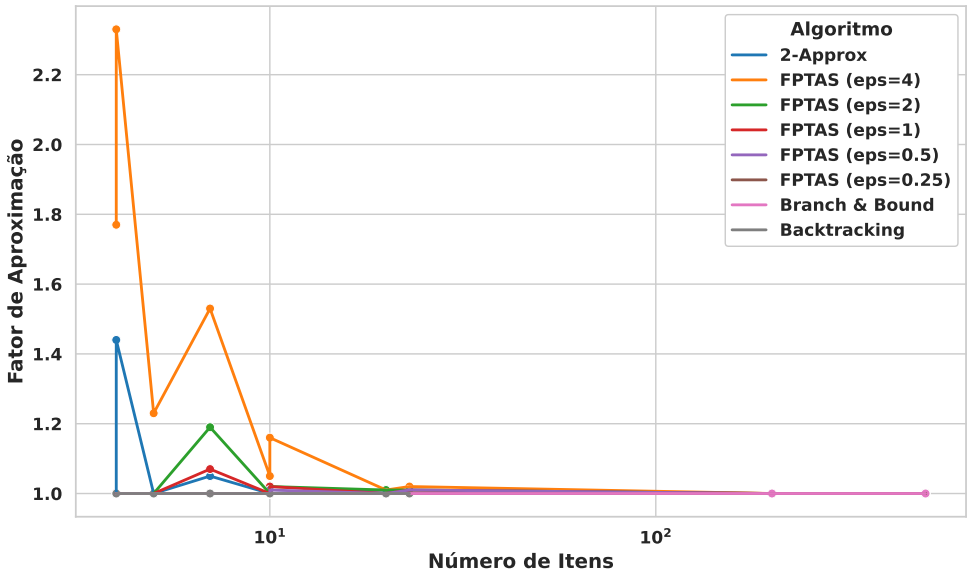

Figure 1. Fator de aproximação em função do número de itens.

Fator de Aproximação: O gráfico do fator de aproximação confirma o esperado para cada abordagem: Branch-and-Bound e Backtracking, quando completos, atingem fator 1 em todas as instâncias resolvidas, validando sua natureza exata. O Novo 2-Aproximativo mantém fator sempre abaixo de 2, com dispersão típica entre 1.0 e 1.2, conforme sua garantia teórica. O FPTAS exibe curvas para diferentes ϵ, aproximando-se de 1 conforme ϵ diminui, evidenciando o controle refinado sobre o trade-off entre qualidade e custo computacional.

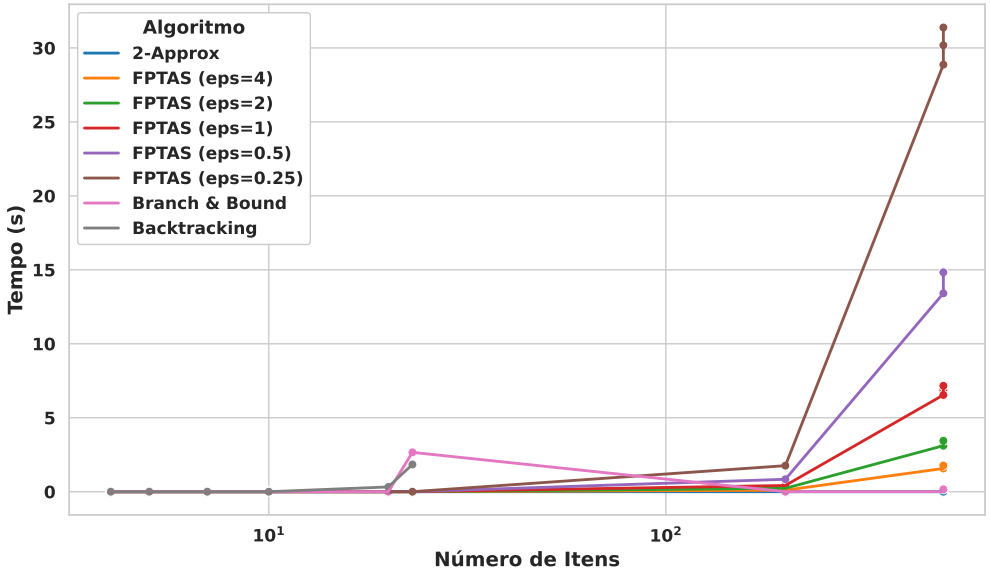

Figure 2. Tempo de execução em função do número de itens.

Tempo de Execução: O gráfico de tempo evidencia a escalabilidade diferente dos algoritmos. O Novo 2-Aproximativo e o Branch-and-Bound apresentam tempos praticamente constantes e mínimos em toda a faixa de tamanhos. Backtracking ultrapassa o limite de tempo para instâncias grandes, confirmando complexidade exponencial. Já o FPTAS mostra curvas distintas para cada ϵ: valores menores resultam em tempo significativamente maior, refletindo complexidade polinomial em $1/\epsilon$.

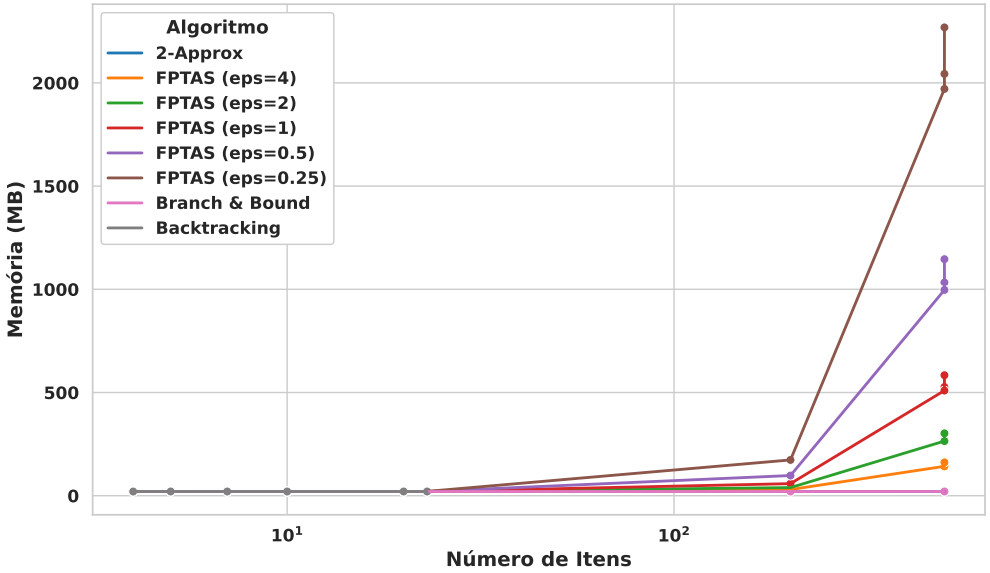

Figure 3. Consumo de memória em função do número de itens.

Uso de Memória: Em termos de memória, os gráficos confirmam o esperado. O Novo 2-Aproximativo e os métodos exatos (Branch-and-Bound, Backtracking) apresentam consumo essencialmente linear com o número de itens, refletindo implementações leves. O FPTAS destaca-se por aumento pronunciado para ϵ menores, evidenciando a complexidade quadrática em n e inversamente proporcional a ϵ, devido ao tamanho das tabelas de programação dinâmica.

3.5. Discussão Geral dos Resultados

A apresentação consolidada dos resultados revela claramente os trade-offs de cada abordagem. O Novo 2-Aproximativo destaca-se pela simplicidade, baixo custo computacional e fator de aproximação controlado. O FPTAS demonstra flexibilidade via ϵ, permitindo aproximações muito próximas do ótimo ao custo de maior tempo e memória para configurações mais exigentes. Branch-and-Bound pode ser viável a depender da instância e Backtracking é viável apenas em instâncias pequenas.

A comparação unificada em gráficos reforça a importância de escolher o algoritmo conforme os requisitos de qualidade, tempo disponível e recursos computacionais. Essa análise conjunta permite visualizar claramente as regiões onde cada técnica é mais indicada e os limites práticos de cada estratégia.

4. Conclusões

Os experimentos evidenciaram claramente o trade-off entre qualidade de solução, tempo de execução e uso de memória ao resolver o Problema da Mochila Binária com diferentes abordagens.

Branch-and-Bound e Backtracking confirmaram eficácia para solução ótima. Mesmo sem ordenação, Branch-and-Bound foi rápido em quase todas as instâncias, inclusive grandes, com tempos comparáveis ao Novo 2-Aproximativo. Isso sugere instâncias favoráveis (bounds fortes, baixa variabilidade), permitindo podas eficazes. No entanto, o pior caso permanece exponencial, tal qual o Backtracking puro que atingiu o timeout de 30 minutos em instâncias grandes, sendo essencial empregar controle de tempo e considerar heurísticas ou aproximações.

Novo 2-Aproximativo demonstrou eficiência notável em tempo e memória. Sua simplicidade resolveu todas as instâncias — pequenas e grandes — em tempos quase nulos e com uso mínimo de recursos. Embora não garanta ótimo absoluto, o fator de aproximação variou entre 1.0 e 1.2. Para aplicações que exigem resultados rápidos com erro controlado (20% no pior caso), o 2-Approx é uma escolha prática e confiável.

FPTAS apresentou comportamento flexível e ajustável. Para valores maiores de ϵ (4.00 ou 2.00), obteve soluções idênticas ou muito próximas do ótimo em várias instâncias, inclusive grandes, com custos controlados. Entretanto, reduzir ϵ para 0.50 ou 0.25 elevou bastante tempo e memória — RAM acima de 2GB e tempos além de 30s em instâncias grandes. Isso confirma sua complexidade polinomial em $1/\epsilon$ e destaca a importância de escolher bem ϵ para evitar custos excessivos.

Síntese geral e recomendações:

- **Branch-and-Bound**: Viável para se obter o ótimo dependendo da instância.
- **Backtracking**: Limitado pelo crescimento exponencial, atingiu timeout em instâncias grandes. Viável apenas para instâncias pequenas.
- **2-Approx**: Ideal para respostas quase imediatas com custo mínimo de memória e fator até 2. Excelente para conjuntos grandes ou sistemas com poucos recursos.
- **FPTAS**: Versátil e ajustável via ϵ, permite controlar o trade-off qualidade/custo. Valores grandes de ϵ ofereceram soluções ótimas ou quase ótimas em muitas instâncias grandes. No entanto, a análise de uso de memória e tempo evidenciou crescimento polinomial em $1/\epsilon$, exigindo escolher ϵ com cuidado para balancear precisão e custo.

Considerações finais: Não há uma abordagem universalmente superior para o Problema da Mochila Binária. A escolha depende do tamanho da instância, restrições de tempo e memória e do grau de precisão necessário. Este trabalho reforça a importância de analisar bem o problema-alvo para selecionar a técnica mais apropriada, demonstrando na prática conceitos teóricos de algoritmos exatos e aproximativos para problemas NP-difíceis.