# Trabalho Prático II - Algoritmos para problemas difíceis
# DCC 207 - Algoritmos II

**Augusto Guerra de Lima, Heitor Gonçalves Leite**

[1]Departamento de Ciência da Computação – Instituto de Ciências Exatas
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil – Inverno de 2025

{augustoguerra,heitorleite}@dcc.ufmg.br

***Abstract.*** *This work investigates algorithms for hard problems, in particular, the knapsack problem. Through an experimental study of three approaches—branch and bound, a fully polynomial-time approximation scheme (FPTAS), and a* 2*-approximation algorithm—this study aims to understand the characteristics of each algorithm regarding runtime, space complexity, and solution quality, under both small and large workloads.*

***Resumo.*** *Este trabalho investiga algoritmos para problemas difíceis, em particular, o problema da mochila. Por meio de um estudo experimental de três abordagens—branch-and-bound, fully polynomial-time approximation scheme (FPTAS) e um algoritmo* 2*-aproximado—busca-se entender o que caracteriza cada um dos algoritmos em termos de tempo de execução, complexidade espacial e qualidade da solução, dadas cargas de trabalho pequenas e grandes.*

## 1. Introdução

Problemas algorítmicos para os quais não se conhece uma solução determinística em tempo polinomial muitas vezes não podem ser simplesmente ignorados, já que podem ser de interesse prático. Algumas abordagens visam mitigar o alto tempo de execução, seja explorando o espaço de solução de forma sistemática como o *branch-and-bound*, seja encontrando soluções boas, mas não ótimas, algoritmos aproximativos.

Neste trabalho, pretende-se investigar tais abordagens, o problema algorítimico escolhido foi o problema da mochila. Para isso, são implementados o algoritmo *branch-and-bound*, o *fully polynomial-time approximation scheme*, e um algoritmo 2-aproximativo a ser discutido. A seção seguinte tratará da metodologia empregada; em seguida, as análises comparativas entre os algoritmos implementados, com a discussão dos resultados obtidos; por fim, serão apresentadas as conclusões e as referências utilizadas.

## 2. Metodologia

Nessa etapa serão discutidos os detalhes das implementações. Os três *solvers* compartilham de uma interface.

### 2.1. Branch-and-bound

O algoritmo de *branch-and-bound* para o problema da mochila pretende resolver o problema de forma exata, isto é, não se trata de um algoritmo aproximativo. Contudo, ele não resolve o problema de forma ingênua; é bem verdade que ele se baseia em uma estratégia de cota superior que é atualizada a cada etapa da busca pela resposta ótima. A **cota superior** é uma estimativa otimista do melhor valor possível que pode ser atingido a cada etapa da busca no espaço de soluções.

O método de exploração utilizado é o ***best-first search***, que consiste em visitar primeiro estados cuja cota superior é maior, em busca de uma solução que *maximize* o valor obtido no problema da mochila. A estrutura de dados utilizada para a implementação dessa abordagem é a **fila de prioridade**; a escolha dessa busca consiste em uma tentativa de mitigar o tempo de execução ao visitar estados mais promissores primeiro.

Finalmente, em mais detalhes, cada **estado** representa um vértice na árvore do espaço de solução do problema da mochila e é definido como:

```
struct BnBSolver::State {
    ull capacity, value, idx, upper_bound = 0;

    bool operator <(const State& other) const {
    return upper_bound > other.upper_bound;
    }
};
```

Encapsulando, através de variáveis do tipo `unsigned long long`, quanto ainda cabe na mochila, o valor corrente dos itens escolhidos, o índice do próximo item a ser considerado na busca e a cota superior, respectivamente.

Já a estimativa é um relaxamento fracionário do problema da mochila; ordenando os itens pela razão valor/peso, `v/w`, que revela o caráter fracionário da cota, obtém-se por

$$Ub = V + (W - w) \cdot \frac{v_{i+1}}{w_{i+1}}$$

a cota superior de um estado $Ub$, que é a soma do valor acumulado no estado e do produto entre a capacidade restante, pela razão valor/peso do item seguinte.

Em suma, a abordagem de *branch-and-bound* busca, no espaço de soluções, estados com maiores cotas superiores e atinge uma resposta quando não há estimativa que supere a folha encontrada. Embora seja um algoritmo exato, sua complexidade de tempo de execução é meramente *pseudopolinomial* e, em particular, para algumas instâncias, o algoritmo degenerar-se-á de forma a varrer toda a árvore de soluções.

### 2.2. Fully polynomial-time approximation scheme

O esquema de aproximação plenamente polinomial é um algoritmo aproximado. O interessante é que o fator de aproximação é parametrizado por $1 > \varepsilon > 0$; tem-se, pois, um algoritmo $(1 - \varepsilon)$-aproximativo. Contudo, é esperado que o tempo de execução aumente para algoritmos mais precisos.

Primeiramente, o algoritmo aplica uma **transformação** na instância de entrada:

$$\mu = \frac{\varepsilon \cdot v_{\max}}{n} \quad \rightarrow \quad v'_i = \left\lfloor \frac{v_i}{\mu} \right\rfloor .$$

O valor reduzido dos itens é utilizado durante a técnica de **programação dinâmica**, garantindo que o tempo de execução seja polinomial. Não obstante, a transformação realiza um truncamento ao arredondar para o menor inteiro mais próximo, o que torna a resposta uma aproximação. Ademais, o valor $\mu$ é uma função $f(\varepsilon)$ que garante um algoritmo com fator de aproximação $(1 - \varepsilon)$.

O estado da programação dinâmica

```
const ull inf = inst.capacity + 1;
std::vector dp(inst.n+1, std::vector<ull>(sumv+1, 0));
```

é definido como, $\text{dp}[i][j] :=$ "O menor peso necessário para somar o valor $j$ utilizando apenas os itens indexados entre $0$ e $i$". Descrito a partir da relação de recorrência:

$$\text{dp}[i][j] = \begin{cases} 0, & \text{se } j = 0; \\ \infty, & \text{se } i = 0 \text{ e } j > 0; \\ \text{dp}[i-1][j], & \text{se } v'_i > j; \\ \min\left(\text{dp}[i-1][j],\ \text{dp}[i-1][j - v'_i] + w_i\right), & \text{se } v'_i \leq j. \end{cases}$$

A implementação foi realizada com a abordagem *bottom-up*, que constrói a tabela de memoização por completo de forma iterativa e, ao contrário da abordagem recursiva, evita empilhar muitas chamadas.

### 2.3. Algoritmo 2-aproximativo

O algoritmo 2-aproximativo garante que a resposta será pelo menos metade do valor ótimo, isto é, seja $C^*$ a resposta exata e $C$ a aproximada, então $0 \leq C^*/C \leq 2$; isso será importante durante a análise experimental.

O funcionamento do algoritmo consiste em, primeiramente, filtrar o subconjunto $S$ de itens onde $S = \{w \mid w \leq W\}$; após isso, estabelece-se uma ordem total no conjunto $S$ de acordo com a razão valor/peso. Consequentemente, uma técnica **gulosa** é aplicada: os itens são inseridos na solução na medida em que a soma dos pesos não ultrapasse a capacidade da mochila.

**Clama-se que:** O algoritmo supracitado tem fator de aproximação 2.

*Prova:* Seja $C^*$ o valor da solução ótima e $C$ o valor retornado pelo algoritmo proposto.

Tome $i$, o primeiro item que não pode ser incluso no conjunto resposta ao empregar a técnica gulosa.

Defina $X = \sum_{v_i \in S' \subseteq S} v_i$ o valor dos itens escolhidos pelo algoritmo guloso, tal que todos couberam na mochila; e defina $Y$ o valor do primeiro item que não coube.

Note que $C = \max\{X, Y\}$; isto é, o algoritmo retorna o máximo entre o total dos itens que couberam $X$ e o valor do primeiro item que não coube $Y$.

Está claro que, como os itens são escolhidos por ordem de valor específico [3], e seja o primeiro item a ultrapassar a capacidade, segue que $C^* \leq X + Y$.

Como $C = \max\{X, Y\}$ então, $2C \geq X + Y \geq C^*$. Portanto:

$$\frac{C^*}{C} \leq 2.$$

□

### 2.4. Complexidade

A complexidade assintótica do algoritmo de *branch-and-bound*, apresentado em [2], é ainda $\mathcal{O}(2^n)$, embora o algoritmo tenha uma grande otimização quando comparado ao de força bruta. Já o esquema de aproximação visto em [1] tem um algoritmo de programação dinâmica $\mathcal{O}(nV)$ que, devido às transformações de valores, torna-se $\mathcal{O}\left(\frac{n^3}{\varepsilon}\right)$. Por fim, o algoritmo 2-aproximativo proposto tem o maior custo na ordenação do conjunto $S$ e, por isso, tem complexidade de tempo de execução em $\mathcal{O}(|S| \cdot \log |S|)$.

## 3. Análise experimental e comparativa

## 4. Conclusões

Neste trabalho foram implementadas três abordagens para a solução do problema da mochila.

## 5. Referências

## References


[1] KLEINBERG, Jon & TARDOS, Éva. *Algorithm Design*. Addison-Wesley, 2005.

[2] LEVITIN, Anany. *Introduction to The Design and Analysis of Algorithms*. 3rd edition. Addison-Wesley, 2012.

[3] *Mochila booleana aproximada*. IME-USP. Acesso em : 04 de julho de 2025; Disponível em: `https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-aprox.html`.