Problema da Mochila 0/1 Utilizando Programação Dinâmica

Eduardo H. F. Machado, *Discente, UFRR - DCC*
Fernando S. Rodrigues, *Discente, UFRR - DCC*

***Resumo*—Esse artigo explora soluções ao Problema da Mochila 0/1 utilizando *Backtracking* e Programação Dinâmica. Ele discute trabalhos relacionados, delineia a metodologia, apresenta resultados de *benchmark* e compara a performance dos algoritmos. O objetivo é entender a eficiência da Programação Dinâmica como técnica para lidar com esse problema NP-difícil e sua performance em relação a outras abordagens como *Backtracking*.**

***Palavras-chave*—Problema da Mochila 0/1, Programação Dinâmica, *Backtracking*, NP-completo.**

***Abstract*—This article explores solutions to the Knapsack Problem 0/1 using Backtracking and Dynamic Programming. It discusses related works, outlines the methodology, shows benchmark results, and compares the performance of the algorithms. The objective is to understand the efficiency of Dynamic Programming as a technique to deal with this NP-complete problem when compared against other approaches, such as Backtracking.**

***Keywords*—Knapsack Problem 0/1, Dynamic Programming, *Backtracking*, NP-complete.**

I. Introdução

O problema da Mochila 0/1 é um dos problemas clássicos de otimização combinatória, frequentemente utilizado como modelo em situações onde há necessidade de alocar recursos limitados de maneira eficiente. O problema consiste em decidir quais itens, de um conjunto, devem ser selecionados para maximizar o valor total, sem exceder uma capacidade máxima imposta por uma restrição de peso. Esse problema tem aplicações em várias áreas, como logística, finanças e ciência da computação. Neste trabalho, abordamos a solução do problema utilizando duas abordagens principais: *Backtracking* e Programação Dinâmica, com o objetivo de avaliar o desempenho de cada uma delas em termos de eficiência computacional e qualidade da solução.

A. Problema de Pesquisa

O Problema da Mochila 0/1 (*Knapsack Problem 0/1*) é um problema de otimização combinatória onde, dada uma coleção de itens, cada um com seu peso e valor, busca-se maximizar o valor total dos itens que serão incluídos na mochila sem exceder a sua capacidade. O problema pode ser formulado como:

Eduardo H. F. Machado and Fernando S. Rodrigues are with Universidade Federal de Roraima, with the Computer Science Department.

$$\text{maximizar} \sum_{i=1}^{n} v_i x_i$$

$$\text{sujeito a} \sum_{i=1}^{n} w_i x_i \leq W$$

onde $x_i \in \{0, 1\}$, v_i é o valor do item i, w_i é o peso do item i, e W é a capacidade da mochila.

B. Objetivos

O objetivo deste trabalho é apresentar soluções para o Problema da Mochila 0/1 utilizando as abordagens de *Backtracking* e Programação Dinâmica, comparar os resultados obtidos com aqueles demonstrados nos trabalhos relacionados [1], além de aplicar essas soluções em um cenário de planejamento de missão espacial.

II. Fundamentação Teórica

O Problema da Mochila 0/1 é um clássico da teoria da complexidade computacional, sendo NP-completo. Ele tem aplicações em diversas áreas, como alocação de recursos, planejamento financeiro e logística. A solução exata pode ser obtida por métodos de força bruta, *Backtracking* e técnicas otimizadas como a Programação Dinâmica.

A. NP-Completude

Para provar que o Problema da Mochila 0/1 é NP-completo, podemos tentar reduzir o Problema da Soma de Subconjuntos, que é NP-completo, ao problema da mochila [2]. Como o problema da mochila booleana é um problema de otimização (busca por um conjunto ótimo), ele precisa ser transformado em um problema de decisão (com resposta ”sim”ou ”não”). Com essa transformação, é possível realizar a redução e, assim, provar que o problema da mochila também pertence à classe NP e é NP-completo .

III. Trabalhos Relacionados

O artigo [1] introduz uma variação chamada Problema da Mochila com Elasticidade de Preços (PEKP), no qual o peso e o valor dos itens dependem do preço. Este problema foi resolvido utilizando programação não linear, com métodos eficientes para casos específicos.

IV. Método

Para resolver o problema da Mochila 0/1, foram implementadas duas abordagens distintas: *Backtracking* e Programação Dinâmica. Tais abordagens que buscam solucionar problemas de otimização combinatória são amplamente utilizado na literatura e nos serviram como base [3].

A abordagem de *Backtracking* consiste em explorar todas as possíveis combinações de itens de forma exaustiva, retornando a solução ótima ao final da busca. Já a programação dinâmica é uma técnica mais eficiente, que resolve o problema ao dividir em subproblemas menores, armazenando as soluções intermediárias para evitar cálculos redundantes [2]. A implementação dessas abordagens foi realizada considerando um cenário fictício de uma missão espacial, onde os cientistas precisam selecionar equipamentos para um *rover* com capacidade limitada.

A. Solução por Backtracking

A solução por backtracking envolve explorar todas as possíveis combinações de itens, verificando em cada estágio se a capacidade da mochila foi excedida. Embora simples, essa abordagem pode ser ineficiente para grandes instâncias do problema devido à sua complexidade exponencial.

A fórmula de recorrência utilizada é:

$$T(n,W) = \begin{cases} 1, & \text{se } n = 0 \text{ ou } W = 0 \\ T(n-1,W), & \text{se } w[n-1] > W \\ 2T(n-1,W)+1, & \text{caso contrário} \end{cases} \tag{1}$$

onde $T(n,W)$ representa o valor máximo que pode ser obtido ao considerar os primeiros n itens com uma capacidade W na mochila, onde $w[i]$ representa o peso do item i.

$$\begin{aligned} T(n,W) &= 2T(n-1,W)+1 \\ T(n-1,W) &= 2[2T(n-2,W)+1]+1 \\ T(n-2,W) &= 2[2^2T(n-3,W)+1]+3 \\ T(k,W) &= 2^kT(n-k,W) + \sum_{i=0}^{k-1} 2^i \\ T(k,W) &= 2^kT(n-k,W) + 2^k - 1 \\ T(n,W) &= 2^{n+1} - 1, \qquad \text{com } k = n \end{aligned} \tag{2}$$

Com base na função de custo obtida (2), temos que a complexidade do algoritmo de *Backtracking* para o Problema da Mochila 0/1 é $O(2^n)$.

B. Solução por Programação Dinâmica

A programação dinâmica resolve o Problema da Mochila 0/1 de maneira eficiente ao dividir o problema em subproblemas menores. A abordagem utiliza uma tabela para armazenar as soluções parciais, evitando a necessidade de recomputar soluções de subproblemas repetidos.

A fórmula de recorrência obtida [3] é:

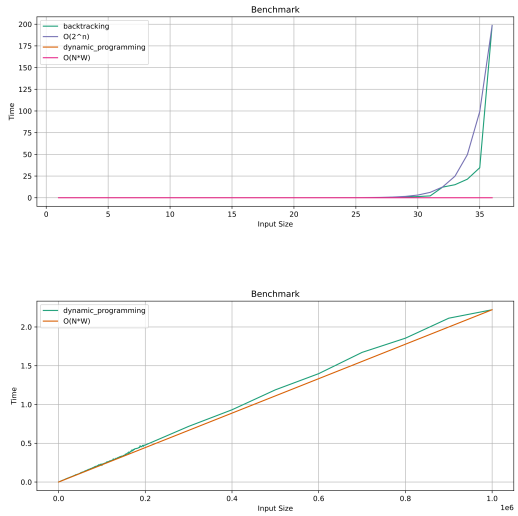

Figura 1. Comparação de Resultados dos *Benchmarks*.

$$T(n,W) = \begin{cases} T(n-1,W), & \text{se } p_n > W \\ \max\{T(n-1,W), T(n-1,W-p_n)+v_n\}, & \text{se } p_n <= W \end{cases} \tag{3}$$

onde $T(n,W)$ representa o valor máximo que pode ser obtido ao considerar os primeiros n itens com uma capacidade W da mochila. A complexidade dessa solução é $O(nW)$.

V. Resultados

Os resultados das implementações foram obtidos com base em *benchmarks* que simulam a seleção de itens para uma missão espacial, conforme descrito na seção de Método. Foram medidos o tempo de execução de cada algoritmo e a solução obtida, comparando o desempenho das duas abordagens. Enquanto o *Backtracking* apresenta maior custo computacional, a Programação Dinâmica obteve a mesma solução ótima em um tempo significativamente menor, o que demonstra a superioridade dessa abordagem em termos de eficiência.

A. Benchmarks

Os benchmarks foram realizados utilizando uma lista de itens com pesos e valores definidos para um cenário de missão espacial:

- Câmera de alta resolução: Peso: 20 kg, Valor: 40 pontos;
- Braço robótico: Peso: 50 kg, Valor: 100 pontos;
- Analisador de solo: Peso: 30 kg, Valor: 60 pontos;
- Detector de radiação: Peso: 10 kg, Valor: 30 pontos;
- Fonte de energia extra: Peso: 40 kg, Valor: 70 pontos;

A capacidade máxima da mochila (*rover*) foi definida como 100 kg.

B. Comparação dos Resultados

Os tempos de execução e a eficiência das soluções de backtracking e programação dinâmica foram comparados.

A solução por backtracking, apesar de encontrar a solução ótima, apresentou desempenho significativamente inferior em termos de tempo de execução em comparação à programação dinâmica, como mostrado na Tabela I.

VI. Discussão

A comparação dos benchmarks demonstra a eficiência da Programação Dinâmica em relação ao *Backtracking*. Para instâncias maiores, a diferença de desempenho tende a aumentar significativamente, tornando o *Backtracking* inviável para aplicações práticas.

VII. Conclusão

Este trabalho apresentou duas abordagens para resolver o Problema da Mochila 0/1. A solução por Programação Dinâmica foi mais eficiente e escalável. Aplicando essas técnicas ao cenário de planejamento de missão espacial, foi possível determinar quais itens maximizaram o valor científico da missão sem exceder o limite de peso do *rover*.

Apêndice A
Materiais Externos

- Código-fonte
- Entradas utilizadas nos *benchmarks*

Referências

[1] D. A. d. O. Ricardo Fukasawa, Joe Naoum-Sawaya, "The price-elastic knapsack problem," *FREEDOM*, vol. 124, pp. 103 003–103 003, 2024. [Online]. Available: https://doi.org/10.1016/j.omega.2023.103003

[2] R. B. Caldas, "Projeto de análise de algoritmos," *DCC-UFMG*, 2004. [Online]. Available: https://homepages.dcc.ufmg.br/~nivio/cursos/pa04/tp2/tp22/tp22.pdf

[3] R. Carvalho, "Problema da mochila," *Universidade Estadual de Campinas Instituto de Matemática, Estatística e Computação Cientifica*, 2015. [Online]. Available: https://www.ime.unicamp.br/~mac/db/2015-1S-122181-1.pdf

Tabela I
RESULTADOS DOS BENCHMARKS (BT = *Backtracking* E PD = PROGRAMAÇÃO DINÂMICA)

Entrada	Solução	Tempo BT (ms)	Tempo PD (ms)
1	23.0	5.38×10^{-7}	4.08×10^{-6}
2	55.0	7.69×10^{-7}	7.08×10^{-6}
3	58.0	1.00×10^{-6}	9.54×10^{-6}
4	110.0	6.92×10^{-7}	1.33×10^{-5}
5	69.0	1.46×10^{-6}	1.58×10^{-5}
6	117.0	1.85×10^{-6}	2.65×10^{-5}
7	165.0	9.23×10^{-7}	2.82×10^{-5}
8	231.0	2.77×10^{-6}	2.16×10^{-5}
9	180.0	2.31×10^{-6}	2.61×10^{-5}
10	335.0	2.23×10^{-6}	2.73×10^{-5}
11	275.0	3.08×10^{-6}	2.85×10^{-5}
12	357.0	1.24×10^{-5}	3.00×10^{-5}
13	197.0	1.48×10^{-5}	3.42×10^{-5}
14	382.0	1.98×10^{-5}	3.84×10^{-5}
15	385.0	3.57×10^{-5}	3.57×10^{-5}
16	312.0	2.42×10^{-4}	3.68×10^{-5}
17	409.0	2.40×10^{-4}	5.02×10^{-5}
18	456.0	3.38×10^{-4}	6.15×10^{-5}
19	388.0	5.25×10^{-4}	5.38×10^{-5}
20	414.0	2.97×10^{-3}	5.19×10^{-5}
21	570.0	3.56×10^{-3}	5.57×10^{-5}
22	623.0	4.86×10^{-3}	6.91×10^{-5}
23	606.0	7.63×10^{-3}	6.52×10^{-5}
24	603.0	4.35×10^{-2}	5.82×10^{-5}
25	758.0	5.22×10^{-2}	5.85×10^{-5}
26	644.0	7.88×10^{-2}	6.12×10^{-5}
27	717.0	1.25×10^{-1}	7.38×10^{-5}
28	710.0	7.46×10^{-1}	6.58×10^{-5}
29	795.0	9.17×10^{-1}	6.92×10^{-5}
30	641.0	1.35	9.56×10^{-5}
31	938.0	2.14	9.88×10^{-5}
32	697.0	1.22×10^{1}	9.56×10^{-5}
33	794.0	1.50×10^{1}	1.01×10^{-4}
34	930.0	2.14×10^{1}	1.22×10^{-4}
35	813.0	3.46×10^{1}	1.01×10^{-4}
36	948.0	1.99×10^{2}	1.31×10^{-4}