Alison Vinicius Soares Martins

Problema da mochila multidimensional utilizando normas vetoriais

Campo Grande, MS

2024

Alison Vinicius Soares Martins

Problema da mochila multidimensional utilizando normas vetoriais

Trabalho de conclusão de curso apresentado a Faculdade de Computação da Universidade de Federal de Mato Grosso do Sul para obtenção do Título de Engenheiro de Computação.

Universidade Federal de Mato Grosso do Sul – UFMS

Faculdade de Computação – FACOM

Orientador: Prof. Dr. Henrique Mongelli

Campo Grande, MS

2024

Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.

Agradecimentos

Primeiramente, agradeço aos meus amigos, que sempre estiveram ao meu lado, oferecendo apoio, companhia e momentos de alegria. Sem vocês, esta jornada teria sido muito mais difícil. Ao meu orientador, agradeço por seu inestimável apoio e orientação. Seus conselhos não apenas me guiaram na pesquisa científica, mas também na vida, ajudando-me a crescer como profissional e como pessoa.

Não posso deixar de expressar minha profunda gratidão às três mães que tive em minha vida: minha mãe biológica, minha tia e minha avó. Cada uma de vocês, com seu amor incondicional, cuidado e sabedoria, moldou quem sou hoje. Ao meu pai, agradeço por seus conselhos certeiros sobre a vida. Sua sabedoria e apoio constante foram fundamentais para que eu pudesse superar os desafios e continuar avançando. Por fim, agradeço à vida por me proporcionar esta jornada única e enriquecedora. Cada experiência, seja boa ou ruim, contribuiu para o meu crescimento e aprendizado.

Resumo

O problema da mochila é um desafio central em otimização combinatória, sendo amplamente aplicável em áreas como logística, economia e ciência da computação. Este problema envolve a seleção de itens para preencher uma mochila com capacidade limitada, com o objetivo de maximizar o valor total dos itens escolhidos. A programação dinâmica é uma abordagem eficiente para resolver este problema, no entanto, sua aplicação direta encontra dificuldades significativas em casos multidimensionais devido à elevada complexidade computacional. Para enfrentar esses desafios, uma abordagem é o uso de listas em algoritmos de programação dinâmica, o que ajuda a gerenciar as combinações de itens. Além disso, a aplicação da norma vetorial é utilizada para comparar as combinações de itens, contribuindo para a redução da complexidade do problema. Essas soluções melhoram a otimização e a eficiência computacional, permitindo encontrar soluções para instâncias mais complexas do problema da mochila.

Palavras-chave: Problema da mochila. Otimização combinatória. Programação dinâmica. Complexidade computacional. Norma vetorial.

Lista de tabelas

Sumário

1 Introdução

O problema da mochila é um dos problemas encontrados em otimização combinatória, podendo ser aplicado em diversas áreas como, por exemplo, logística, empacotamento de cargas e em economia. O problema se dá na seleção de itens, cada um com respectivo valor e peso, que serão adicionados a uma mochila com determinada capacidade. O objetivo é maximizar o valor do conjunto de itens selecionados que respeite a capacidade máxima da mochila.

Como este problema possui complexidade exponencial, diversas técnicas foram desenvolvidas para encontrar soluções ótimas. Uma dessas técnicas é a programação dinâmica, desenvolvida por Bellman (1957) (BELLMAN, 1957), que quebra o problema em subproblemas menores resolvendo-os uma única vez e armazenando as soluções. A utilização de programação dinâmica no problema da mochila possui desvantagem quando o número de itens e a capacidade da mochila aumentam consideravelmente, tornando o uso desta técnica impraticável devido ao elevado uso de espaço e tempo.

Ao adicionar restrições ao problema como múltiplas capacidades, a complexidade computacional cresce ainda mais. Esta adição de restrições é dada pela variante multidimensional do problema da mochila. Neste caso, a programação dinâmica se torna ainda mais inviável devido ao crescimento exponencial de acordo com a quantidade de restrições. Abordagens alternativas envolvendo o uso de heurísticas e meta-heurísticas, como algoritmos genéticos, busca tabu e algoritmos de colônia de formigas, que, embora não garantam uma solução ótima, podem encontrar soluções boas em tempo aceitável.

Neste trabalho, foi explorado o uso de normas vetoriais para a diminuição do tempo de execução e consumo de espaço utilizados pelo problema da mochila multidimensional. A ideia da utilização de normas vetoriais é agregar múltiplas restrições em um único valor, simplificando a análise e a resolução do problema.

Ao utilizar normas vetoriais, é possível agregar múltiplas restrições em uma única função de custo, simplificando a análise e a resolução do problema. Essa técnica não só facilita a implementação computacional, mas também pode melhorar a convergência das soluções, oferecendo uma maneira prática de lidar com a alta dimensionalidade e as restrições complexas inerentes ao problema da mochila.

2 Contextualização

Dado um conjunto de n itens e uma mochila com capacidade M, o objetivo é maximizar o valor total dos itens carregados na mochila, respeitando sua capacidade. Este problema pode ser formulado da seguinte maneira:

$$\begin{aligned} \text{maximizar} \quad & \sum_{i=1}^{n} v_i x_i \\ \text{tal que} \quad & \sum_{i=1}^{n} w_i x_i \leq M \qquad x_i \in \{0,1\} \quad i = 1, \ldots, n \end{aligned}$$

Neste contexto, v_i representa o valor do item i, w_i é o peso do item i, M é a capacidade máxima da mochila e x_i é uma variável binária que indica se o item i será selecionado ($x_i = 1$) ou não ($x_i = 0$).

Este modelo matemático captura a essência do problema da mochila 0-1, equilibrando o valor total dos itens selecionados com a restrição de capacidade da mochila. A solução ideal requer escolher uma combinação de itens que maximize o valor total sem ultrapassar a capacidade disponível.

Enquanto o problema da mochila 0-1 tem apenas uma restrição de capacidade, o problema da mochila multidimensional apresenta múltiplas restrições de capacidade, $M = \{M_1, \ldots, M_d\}$. Isso resulta em um aumento da quantidade de pesos associados a cada item. A formulação matemática do problema da mochila multidimensional pode ser representada da seguinte maneira:

$$\begin{aligned} \text{maximizar} \quad & \sum_{i=1}^{n} v_i x_i \\ \text{tal que} \quad & \sum_{i=1}^{n} w_{ij} x_i \leq M_j, \qquad j = 1, \ldots, d \\ & x_i \in \{0,1\} \qquad i = 1, \ldots, n \end{aligned}$$

sendo v_i o valor do item i, w_{ij} o peso do item i na dimensão j, M_j a capacidade máxima na dimensão j, e x_i uma variável binária que indica se o item i será ou não selecionado.

Existem várias abordagens para encontrar soluções para o problema da mochila 0-1. Duas estratégias principais são *branch and bound* e a técnica de programação dinâmica (KELLERER; PFERSCHY; PISINGER, 2004). O primeiro é exponencial, pois considera todas as combinações possíveis, resultando em complexidade $O(2^n)$. Já o segundo é considerado pseudopolinomial,

pois leva em consideração o tamanho do conjunto de itens e o tamanho da mochila, ou seja, possui complexidade $O(nM)$. Existem variações do algoritmo de programação dinâmica que visam melhorar o tempo de execução ou reduzir o consumo de espaço, como, por exemplo, o algoritmo de programação dinâmica com listas (KELLERER; PFERSCHY; PISINGER, 2004, p. 71), que mantém a complexidade em tempo pseudopolinomial enquanto reduz a quantidade de espaço utilizado.

Quando lidamos com a variante multidimensional do problema da mochila 0-1, em que há múltiplas restrições de capacidade, o tempo de execução do algoritmo de programação dinâmica pode se tornar impraticável. Isso ocorre devido à sua complexidade exponencial $O(nM_1...M_d)$, especialmente quando o número de dimensões, d, é grande. Para contornar esse desafio, recorremos a heurísticas e meta-heurísticas, como *Simulated Annealing* (KIRKPATRICK; JR.; VECCHI, 1983), *Tabu Search* (VARNAMKHASTI, 2012) e Algoritmos Genéticos (HAUPT; HAUPT, 2003) . Essas abordagens são capazes de encontrar soluções aproximadas de forma mais eficiente em termos de tempo de execução. No entanto, é importante ressaltar que essas soluções não oferecem garantia de exatidão, embora sejam frequentemente boas em muitos casos.

3 Programação dinâmica

Desenvolvida na década de 1950 pelo matemático Richard Bellman (BELLMAN, 1957), a programação dinâmica se tornou uma ferramenta poderosa para resolver problemas computacionais. A ideia por trás da programação dinâmica é a divisão do problema em subproblemas menores, que são resolvidos individualmente e combinados para chegar à solução do problema. Devido a essas características, é amplamente utilizada em otimização combinatória, na qual o objetivo é maximizar ou minimizar uma função.

O problema da mochila pode ser resolvido com a utilização da programação dinâmica (KELLERER; PFERSCHY; PISINGER, 2004) de diferentes maneiras.Um dos algoritmos mais conhecidos identifica a estrutura dos subproblemas, ou seja, divide o problema em partes menores, simplificando a busca pela solução. Para maximizar a função objetivo do problema da mochila, é necessário utilizar uma matriz Z com n linhas e M colunas, e cada posição da matriz representa o valor máximo possível que pode ser alcançado em determinado subproblema.

Para o cálculo de cada posição da matriz, é levada em consideração a relação:

$$Z_{ij} = \begin{cases} Z_{i-1j} & \text{se } j < w_i \\ \max\{Z_{i-1j}, Z_{i-1j-w_i} + v_i\} & \text{se } j \geq w_i \end{cases}$$

Essa relação demonstra que, se a coluna de Z tem tamanho menor que o peso do item i, então o valor da posição acima é copiado para a posição atual, caso contrário, é encontrado o valor máximo entre a posição acima e a posição anterior calculada a partir do peso do item e somada com o valor do item. O algoritmo que utiliza esta relação é o Algoritmo 1.

Com a matriz Z preenchida a partir do Algoritmo 1, para encontrar os itens que geraram a solução exata, basta utilizar o Algoritmo 2.

O vetor S tem tamanho n, correspondente ao número de itens. O Algoritmo 2 percorre a matriz a partir da posição $Z[n][W]$, enquanto $i > 0$ e $j \geq 0$. Ao encontrar um valor em $Z[i][j]$ que difere de $Z[i-1][j]$, isto é, o item i foi incluído na solução ótima, o algoritmo define o valor de $S[i]$ como 1. Ao final da iteração, o conjunto solução é formado pelos itens cujos índices possuem valor 1 na posição correspondente no vetor S.

Como descrito, existe a variação do Algoritmo 1 que utiliza listas em sua execução. Inicialmente, o Algoritmo 3 começa com uma lista vazia possuindo uma combinação (v, w) vazia, ou seja, valor e peso iguais a zero. Em seguida, é percorrido o conjunto de itens dado na entrada, realizando todas as combinações possíveis do item i com cada célula contida na lista L. Esta combinação é realizada pela função $Combina(L, (v_i, w_i))$, que soma o valor e o peso do item i com as combinações já presentes na lista L, respeitando a capacidade M da mochila. O

Algoritmo 1: Algoritmo que retorna matriz Z preenchida

Input: Vetor w de pesos, vetor v de valores, capacidade W
Result: Conjunto solução que gerou a solução da matriz Z

```
for j ← 0 to W do
    Z[0][j] ← 0
end
for i ← 1 to n do
    for j ← 0 to W do
        if j ≥ w[i] then
            Z[i][j] ← max(Z[i − 1][j], Z[i − 1][j − w[i]] + v[i])
        end
        else
            Z[i][j] ← Z[i − 1][j]
        end
    end
end
return Z
```

Algoritmo 2: Algoritmo para retornar o conjunto solução para o Problema da Mochila utilizando Programação Dinâmica

Input: Matriz Z
Result: Conjunto solução que gerou a solução da matriz Z

```
i ← n
j ← W
while i > 0 and j ≥ 0 do
    if Z[i][j] ≠ Z[i − 1][j] then
        S[i] = 1
        j ← j − w[i]
    end
    i ← i − 1
end
return S
```

resultado dessa operação é armazenado na lista L'. Após a combinação, é realizado a junção da lista L' com a lista L a partir da função $Merge(L, L')$ seguindo os seguintes casos:

- **Caso w = w'**: A combinação de maior valor é mantida e a de menor valor é excluída.
- **Caso w > w'**: Se $v > v'$, então adiciona a combinação (v, w) em L, caso contrário adiciona a combinação (v', w') em L e exclui-se a combinação (v, w).
- **Caso w < w'**: Se $v > v'$,então adiciona a combinação (v, w) em L e exclui-se (v', w'), caso contrário adiciona (v', w') em L.

O Algoritmo 3 tem a vantagem de diminuir o número de combinações geradas durante a execução, mas ainda mantém tempo de execução pseudopolinomial $O(nC)$, sendo C o número

de combinações geradas pertencentes a lista L.

Algoritmo 3: Algoritmo com Listas para o Problema da Mochila

Input: conjunto de itens, tamanho n
Result: Combinação de maior valor na lista L

```
1  L ← {(0, 0)}
2  for i ← 1 to n do
3  |   L' ← Combina(L, (v_i, w_i))
4  |   L ← Merge(L, L')
5  end
6  return A maior combinação em L_n
```

Algoritmo 4: Algoritmo que realiza a comparação entre duas combinações com d pesos

Input: combinações a e b junto com o tamanho d
Result: flag que identifica se é maior, menor ou igual

```
1  flag ← 0
2  for i ← 1 to d do
3  |   if a_i > b_i then
4  |   |   flag ← 1
5  |   end
6  |   else
7  |   |   if a_i < b_i then
8  |   |   |   flag ← 2
9  |   |   end
10 |   end
11 end
12 return flag
```

Os Algoritmos 1 e 3 podem ser adaptados para o problema da mochila multidimensional. Para o Algorimo 1, a matriz de cálculo Z aumenta a quantidade de colunas conforme o número de dimensões, ou seja, Z possuirá $M_1 M_2 \ldots M_d$ colunas e n linhas, e, na execução que preenche a matriz, serão adicionados d laços de execução. Para o Algoritmo 3, a modificação ocorre na comparação dos pesos, dentro da função MergeLists(L, $L^{'}$). A comparação é descrita no Algoritmo 4.

Esta comparação se aplica somente aos algoritmos derivados do algoritmo de programação dinâmica para o problema da mochila pelo motivo de levar em consideração as posições de cada combinação na matriz Z, ou seja, as combinações estão em ordem crescente de pesos dentro da matriz. Como o Algoritmo 3 é derivado do Algoritmo 1, utilizando listas para armazenar as combinações, o Algoritmo 4 se aplica já que este determina qual combinação é maior, menor ou igual observando somente os pesos.

Uma vantagem do Algoritmo 3 com relação ao Algoritmo 1 é o consumo de espaço pelo motivo de que será gerada somente uma única combinação a partir das combinações anteriores. As condições aplicadas dentro da função $MergeLists$ excluem elementos da lista que não

serão utilizados em combinações futuras e mantém a ordem de acordo com os pesos de cada combinação. Porém, no pior caso o consumo de espaço, ao final da execução, é equivalente ao espaço utilizado pela matriz Z.

Ambos algoritmos adaptados para o problema da mochila multidimensional possuem complexidade de tempo $O(n.M_1.M_2.\ldots.M_d)$, tornando muitas instâncias inviáveis de serem executadas pelo tempo elevado de execução.

4 Programação dinâmica com listas utilizando norma vetorial

A norma vetorial é uma ferramenta fundamental em Matemática, principalmente no campo da Álgebra Linear. Esta ferramenta permite a medição do tamanho ou magnitude de um vetor em um determinado espaço vetorial. Existem diferentes tipos de normas vetoriais, como, por exemplo, norma 1, norma 2 e norma infinita.

Existe uma generalização para norma vetorial, chamada de norma p. Esta norma é calculada somando-se as componentes do vetor elevadas à potência p, em seguida é tirada a raiz p-ésima da soma. A norma vetorial $||\mathbf{v}||_p$ é definida como:

$$||\mathbf{v}||_p = (|v_1|^p + |v_2|^p + \cdots + |v_n|^p)^{\frac{1}{p}} \tag{4.1}$$

sendo $\mathbf{v} = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}$ um vetor com n componentes e p um parâmetro que determina o tipo de norma a ser calculada. Quando $p \to \infty$, a norma vetorial tende para o maior valor contido no vetor, sendo esta a norma infinito. A representação matemática desta norma é:

$$\|\mathbf{v}\|_\infty = \max_i |v_i| \tag{4.2}$$

A norma vetorial possui as seguintes propriedades:

1. **Não negatividade**: $||\mathbf{v}||_p \geq 0$ para todo vetor $\boldsymbol{v}$ em um espaço vetorial, e $||\mathbf{v}||_p = 0$ se e somente se $\boldsymbol{v} = \mathbf{0}$.
2. **Homogeneidade**: $||\mathbf{v}||_p = |\alpha| \cdot ||\mathbf{v}||_p$ para todo escalar α e todo vetor $\boldsymbol{v}$ em um espaço vetorial.
3. **Desigualdade triangular**: $||\mathbf{v} + \mathbf{w}||_p \leq ||\mathbf{v}||_p + ||\mathbf{w}||_p$ para quaisquer vetores $\mathbf{v}$ e $\mathbf{w}$ em um espaço vetorial.

Utilizando as propriedades de não negatividade e homogeneidade, o algoritmo que calcula a norma vetorial do vetor v com tamanho d, é dada pelo Algoritmo 5.

Com o Algoritmo 5, que calcula a norma vetorial, é possível reduzir o consumo de espaço e tempo de execução do algoritmo de programação dinâmica com listas, para o problema da mochila multidimensional, apenas modificando o comparador de pesos dentro da função MergeLists. A modificação proposta é descrita no Algoritmo 6.

Para utilizar a norma vetorial no Algoritmo 6, define-se α como 10^x, sendo x a quantidade de casas decimais desejadas da norma vetorial, e define-se A e B como variáveis inteiras. Assim,

Algoritmo 5: Algoritmo que realiza o cálculo da norma vetorial de um vetor v

Input: vetor v, tamanho d e escalar α
Result: norma vetorial multiplicada pelo escalar α

$norma \leftarrow 0$
for $i \leftarrow 1$ *to* d **do**
  $norma \leftarrow norma + v_i^p$
end
return $\alpha.norma^{\frac{1}{p}}$

Algoritmo 6: Algoritmo que realiza a comparação entre duas combinações com d pesos a partir da norma vetorial

Input: combinações a e b junto com o tamanho d
Result: flag que identifica se a é maior, menor ou igual a b

$A \leftarrow norma(a, p, \alpha)$
$B \leftarrow norma(b, p, \alpha)$
$flag \leftarrow 0$
if $A > B$ **then**
  $flag \leftarrow 1$
end
else
  if $A < B$ **then**
    $flag \leftarrow 2$
  end
end
return $flag$

as comparações seguintes para a variável flag serão comparações de inteiros. Para encontrar a norma infinito, o retorno do Algoritmo 5 será o maior elemento encontrado no vetor de entrada.

Como a norma das d dimensões é $\mathbf{M} = ||M_1 \cdot M_2 \cdot \ldots \cdot M_d||$ e a norma vetorial é menor que o produto $M_1 \cdot M_2 \cdot \ldots \cdot M_d$ para $d > 1$, ou seja, $\mathbf{M} < M_1 \cdot M_2 \cdot \ldots \cdot M_d$, então a complexidade de tempo e espaço é reduzida, tornando-se $O(n \cdot \mathbf{M})$.

5 Metodologia

O algoritmo de programação dinâmica com listas foi executado utilizando como entrada os conjuntos de teste GK, WEING, HP e SENTO, todos coletados no repositório de J. Drake (DRAKE, 2015). Para avaliar o valor da solução encontrada, foi executado o cálculo do erro relativo a partir da equação:

$$E_r = \frac{|V_{\text{calculado}} - V_{\text{referência}}|}{|V_{\text{referência}}|} \times 100$$

O ambiente de teste utilizado para a implementação, compilação e execução de cada algoritmo e os respectivos testes possui a seguinte configuração:

- processador intel i7-9750H.
- 16GB de RAM.
- GPU NVIDIA GTX 1650.
- SSD de 500GB.
- linguagem C++.

Para cada conjunto de testes foram realizadas execuções com p possuindo valores de 2 a 5 e, por último, foi executado com $p \to \infty$. Além destes valores para p, o valor de α foi fixado em 1000 para garantir precisão de 3 casas decimais após a vírgula.

6 Resultados

As Tabelas 1, 2, 3, 4 e 5 apresentam os resultados obtidos a partir da execução de cada teste. Cada tabela possui as colunas Teste, Valor Encontrado, Referência, E_r e Tempo. A primeira coluna contém o nome de cada teste executado, a segunda coluna mostra o valor encontrado para o respectivo teste, a terceira coluna apresenta o valor de referência para o respectivo teste, a quarta coluna exibe o erro relativo entre o valor encontrado e o valor de referência, e a quinta coluna indica o tempo de execução de cada teste em segundos.

Para o teste **hp2**, observa-se que o erro relativo (E_r) diminuiu nas Tabelas 2 e 3, indicando uma melhoria na resposta obtida pelo Algoritmo 3. Porém, mesmo com este comportamento, as Tabelas 4 e 5 demonstram aumento no erro relativo. Em todas as tabelas, o teste **hp2** apresenta maior erro relativo em comparação com os demais testes.

Os testes **weing2**, **weing3**, **weing5** e **weing8** apresentaram erro relativo nulo em todas as tabelas, indicando que o Algoritmo 3 foi capaz de encontrar o valor exato para estes testes. O teste **hp1** apresentou diminuição do erro relativo até a Tabela 4, onde atingiu erro relativo nulo e, na Tabela 5, é observado aumento no erro relativo. O teste **weing1** manteve o erro relativo constante até a Tabela 4 e,na Tabela 5, o erro relativo do teste foi nulo.

Os testes do conjunto GK, possuem tempo elevado de execução devido a quantidade de itens e dimensões, correspondendo à complexidade de tempo descrita anteriormente. Neste mesmo conjunto, o teste **gk09** manteve-se como o teste de maior tempo de execução para todos os valores de p. Observa-se que, ao aumentar o valor de p, o erro relativo tende a diminuir para os testes do conjunto GK, com exceção do caso em que $p \to \infty$, que ocorre um aumento significativo no erro relativo. Os testes deste conjunto apresentam bons resultados em termos de precisão, demonstrando erros relativos menores que 3%, com p de dois a cinco. É observado, também que, para o conjunto GK, o tempo de execução diminuiu conforme o valor de p aumentou. Na Tabela 5 é observado que o tempo de execução para todos os testes diminui consideravelmente em relação às tabelas anteriores.

Para execuções com o valor de $p > 5$, ocorreram limitações de memória devido às potências calculadas para encontrar a norma vetorial. A posição de memória que armazena a variável com este valor não possui tamanho suficiente para acomodar valores grandes.

Todos os tempos de execução encontrados na Tabela 5 são menores em comparação com o tempo encontrado nas tabelas anteriores devido ao cálculo da norma infinito, que retorna o maior valor contido no vetor.

Tabela 1 – Resultados obtidos para a norma $p = 2$

Teste	**Valor Encontrado**	**Referência**	E_r **(%)**	**Tempo (s)**
gk01	3724	3766	1.12	2.250
gk02	3920	3958	0.96	3.920
gk03	5533	5656	2.17	13.180
gk04	5673	5767	1.63	34.530
gk05	7410	7560	1.98	28.990
gk06	7513	7677	2.14	83.110
gk07	19103	19220	0.61	384.980
gk08	18574	18806	1.23	1121.060
gk09	57847	58087	0.41	**7742.370**
weing1	140778	141278	0.35	0.006
weing2	130883	130883	**0**	0.005
weing3	95677	95677	**0**	0.003
weing4	115831	119337	2.94	0.005
weing5	98796	98796	**0**	0.003
weing6	130233	130623	0.30	0.005
weing7	1094460	1095445	0.09	0.148
weing8	624319	624319	**0**	0.028
hp1	3236	3418	5.32	0.012
hp2	2913	3186	**8.57**	0.013
sento1	7761	7772	0.14	0.041
sento2	8711	8722	0.13	0.086

Tabela 2 – Resultados obtidos para a norma $p = 3$

Teste	**Valor Encontrado**	**Referência**	E_r **(%)**	**Tempo (s)**
gk01	3730	3766	0.96	2.144
gk02	3920	3958	0.96	3.690
gk03	5540	5656	2.05	11.907
gk04	5675	5767	1.60	29.582
gk05	7415	7560	1.92	25.483
gk06	7509	7677	2.19	70.030
gk07	19112	19220	0.56	299.601
gk08	18589	18806	1.15	852.677
gk09	57825	58087	0.45	**5938.6**
weing1	140778	141278	0.35	0.006
weing2	130883	130883	**0**	0.006
weing3	95677	95677	**0**	0.003
weing4	115831	119337	2.94	0.004
weing5	98796	98796	**0**	0.003
weing6	130233	130623	0.30	0.006
weing7	1094460	1095445	0.09	0.115
weing8	624319	624319	**0**	0.030
hp1	3364	3418	1.58	0.011
hp2	2994	3186	**6.03**	0.013
sento1	7761	7772	0.14	0.042
sento2	8711	8722	0.13	0.084

Tabela 3 – Resultados obtidos para a norma $p = 4$

Teste	**Valor Encontrado**	**Referência**	E_r **(%)**	**Tempo (s)**
gk01	3743	3766	0.61	1.938
gk02	3920	3958	0.96	3.361
gk03	5543	5656	2.00	10.128
gk04	5677	5767	1.56	26.244
gk05	7446	7560	1.51	22.665
gk06	7515	7677	2.11	62.003
gk07	19115	19220	0.55	254.735
gk08	18577	18806	1.22	709.951
gk09	57836	58087	0.43	**4895.69**
weing1	140778	141278	0.35	0.004
weing2	130883	130883	**0**	0.002
weing3	95677	95677	**0**	0.001
weing4	115831	119337	2.94	0.002
weing5	98796	98796	**0**	0.001
weing6	130233	130623	0.30	0.002
weing7	1094460	1095445	0.09	0.089
weing8	624319	624319	**0**	0.028
hp1	3364	3418	1.58	0.010
hp2	2994	3186	**6.03**	0.012
sento1	7761	7772	0.14	0.040
sento2	8711	8722	0.13	0.082

Tabela 4 – Resultados obtidos para a norma $p = 5$

Teste	**Valor Encontrado**	**Referência**	E_r **(%)**	**Tempo (s)**
gk01	3743	3766	0.61	1.797
gk02	3920	3958	0.96	3.033
gk03	5543	5656	2.00	9.213
gk04	5684	5767	1.44	23.982
gk05	7424	7560	1.80	20.574
gk06	7518	7677	2.07	55.847
gk07	19114	19220	0.55	226.016
gk08	18588	18806	1.16	633.814
gk09	57809	58087	0.48	**4265.08**
weing1	140778	141278	0.35	0.008
weing2	130883	130883	**0**	0.006
weing3	95677	95677	**0**	0.002
weing4	115831	119337	2.94	0.002
weing5	98796	98796	**0**	0.001
weing6	130233	130623	0.30	0.002
weing7	1094460	1095445	0.09	0.084
weing8	624319	624319	**0**	0.028
hp1	3418	3418	**0**	0.009
hp2	2933	3186	**7.94**	0.012
sento1	7761	7772	0.14	0.041
sento2	8711	8722	0.13	0.082

Tabela 5 – Resultados obtidos para a norma $p \to \infty$

Teste	**Valor Encontrado**	**Referência**	E_r **(%)**	**Tempo (s)**
gk01	3611	3766	4.12	0.031
gk02	3809	3958	3.76	0.024
gk03	5439	5656	3.84	0.055
gk04	5571	5767	3.40	0.090
gk05	7314	7560	3.25	0.092
gk06	7475	7677	2.63	0.160
gk07	18646	19220	2.99	0.622
gk08	18295	18806	2.72	1.014
gk09	56574	58087	2.60	**5.478**
weing1	141278	141278	**0**	0
weing2	130883	130883	**0**	0
weing3	95677	95677	**0**	0
weing4	115831	119337	2.94	0.001
weing5	98796	98796	**0**	0
weing6	130233	130623	0.30	0
weing7	1095357	1095445	0.01	0.007
weing8	624319	624319	**0**	0.003
hp1	3292	3418	**3.69**	0.001
hp2	2915	3186	**8.51**	0
sento1	7700	7772	0.93	0.003
sento2	8711	8722	0.13	0.004

7 Considerações Finais

Este trabalho contribuiu com a implementação da norma vetorial como método de comparação de tamanhos no Algoritmo de Programação dinâmica com listas. A partir dos resultados obtidos, observa-se que a norma vetorial é uma ferramenta matemática útil para encontrar soluções boas para o problema da mochila multidimensional. A norma vetorial reduz os pesos das combinações para um único valor, facilitando as comparações realizadas pelo Algoritmo 3.

À medida que o valor de p aumenta, calcular a norma vetorial se torna inviável devido ao tamanho dos valores gerados pelas potências necessárias para o cálculo, este sendo um limitante para obter resultados melhores. Outra desvantagem de utilizar a norma vetorial no Algoritmo 6 é a perda de informações na comparação, pois apenas a norma vetorial de cada combinação será considerada, respeitando o número desejado de casas decimais. Assim, durante a execução do Algoritmo 3, as subsoluções que levam à solução exata podem ser excluídas, fazendo com que o algoritmo retorne apenas uma solução aproximada da exata.

Para projetos futuros, a norma vetorial poderá ser implementada em outras variações do Algoritmo de Programação Dinâmica para o problema da mochila, assim como implementar reduções de consumo de espaço.

Referências

BELLMAN, R. *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957. (Princeton Landmarks in Mathematics and Physics). Revised edition, 2010. ISBN 9780691146683. Disponível em: <https://press.princeton.edu/books/paperback/9780691146683/dynamic-programming>. Citado 2 vezes nas páginas 7 e 10.

DRAKE, J. Benchmark instances for the multidimensional knapsack problem. In: . [s.n.], 2015. Disponível em: <https://www.researchgate.net/publication/271198281_Benchmark_instances_for_the_Multidimensional_Knapsack_Problem>. Citado na página 16.

HAUPT, R. L.; HAUPT, S. E. *Practical Genetic Algorithms*. New York: John Wiley & Sons, Inc., 2003. First published on 14 May 2003. ISBN 9780471455653. Disponível em: <https://doi.org/10.1002/0471671746>. Citado na página 9.

KELLERER, H.; PFERSCHY, U.; PISINGER, D. *Knapsack Problems*. Berlin, New York: Springer, 2004. ISBN 978-3-540-40286-2. Disponível em: <https://lib.ugent.be/catalog/rug01:001052966>. Citado 3 vezes nas páginas 8, 9 e 10.

KIRKPATRICK, S.; JR., C. G.; VECCHI, M. Optimization by simulated annealing. *Science*, v. 220, n. 4598, p. 671–680, May 1983. ISSN 0036-8075. Disponível em: <https://doi.org/10.1126/Science.220.4598.671>. Citado na página 9.

VARNAMKHASTI, M. Overview of the algorithms for solving the multidimensional knapsack problems. *Advanced Studies in Biology*, v. 4, n. 1, p. 37–47, 2012. Citado na página 9.