Exploring Efficient Solutions for the 0/1 Knapsack Problem

Dalal M. Althawadi[1], Sara Aldossary[1], Aryam Alnemari[1], Malak Alghamdi[1], Fatema Alqahtani[1], Atta-ur Rahman[2,*], Aghiad Bakry[2] and Sghaier Chabani[1]

[1]Department of Networks and Communication, College of Computer Science and Information Technology, Imam Abdulrahman Bin Faisal University, P.O. Box 1982, Dammam 31441, Saudi Arabia
[2]Department of Computer Science, College of Computer Science and Information Technology, Imam Abdulrahman Bin Faisal University, P.O. Box 1982, Dammam 31441, Saudi Arabia
[*]Correspondence: aaurahman@iau.edu.sa

Abstract
One of the most significant issues in combinatorial optimization is the classical NP-complete conundrum known as the 0/1 Knapsack Problem. This study delves deeply into the investigation of practical solutions, emphasizing two classic algorithmic paradigms, brute force, and dynamic programming, along with the metaheuristic and nature-inspired family algorithm known as the Genetic Algorithm (GA). The research begins with a thorough analysis of the dynamic programming technique, utilizing its ability to handle overlapping subproblems and an ideal substructure. We evaluate the benefits of dynamic programming in the context of the 0/1 Knapsack Problem by carefully dissecting its nuances in contrast to GA. Simultaneously, the study examines the brute force algorithm, a simple yet comprehensive method compared to Branch & Bound. This strategy entails investigating every potential combination, offering a starting point for comparison with more advanced techniques. The paper explores the computational complexity of the brute force approach, highlighting its limitations and usefulness in resolving the 0/1 Knapsack Problem in contrast to the set above of algorithms.

Keywords:
Dynamic programming, Genetic Algorithms, Brute force, Branch and Bound algorithm, knapsack problem, efficiency

1. Introduction

Despite rigorous advancement in software and hardware resources, the design and analysis of algorithms to find the most efficient one has always been the hottest area of research in optimization. The 0/1 The knapsack problem is a classic optimization problem in computer science, engineering, and combinatorial optimization, with considerable importance in various fields, including operations research, algorithm design, and theoretical computing. This problem can be briefly expressed as follows: given a finite set of items, each of which has a certain weight and value, determine the optimal selection of items to include in a backpack of limited capacity such that the total weight does not exceed the capacity and the total value is maximized [1]. It is a constrained optimization problem that mimics several real-life problems, such as revenue enhancement under a fixed deposit, and hence needs to be solved effectively.

In real life, when we want to solve a problem, we make a set of steps to solve this problem. Also, there are many problems in the technology world, one of them being power consumption. Some devices have a high power consumption, generating more heat. Therefore, companies and enterprises seek to lower the power consumed by their devices, and one of the essential ways is choosing a suitable scheduling algorithm. The software will be implemented to provide different scheduling algorithms to analyze which one is the best [2-3]. This means the CPU will work more efficiently and generate less heat, ensuring a potentially sustainable and renewable solution. This study aims to ensure that these algorithms provide correct results with due efficiency [4]. We are going to use Python programming language in this regard.
This paper is organized as follows. The next section describes the methodology we followed in this study. Section 3 discusses the companies' survey results. In section 4, we discuss the results of the students' survey. Section 5 summarizes the results and gives some recommendations. Finally, in section 6, we give some concluding remarks.

2. Background

Algorithms have become a crucial component of every subject. From the very beginning, a good algorithm ensures great simplifying of things and aids in problem-solving. Sort algorithms are a crucial component of computer science because they offer an organized method of managing and organizing data,

Manuscript received February 5, 2024
Manuscript revised February 20, 2024
https://doi.org/10.22937/IJCSNS.2024.24.2.2

from refining data processing to looking for certain things in a dataset. Merger algorithms are an example of a sorting algorithm, a "divide and conquer" method that splits the input in half recursively, sorts each half separately, and then combines the sorted halves to get the final sorted output [5]. Moreover, the counting algorithm is a non-comparison-based sorting method that performs well when the input value range is constrained by the counting sort [6].

An algorithm is the most crucial element while executing the processes to make sure that the CPU is working at peak speed without degrading its performance with the possible lowest temperature of the CPU; this means the efficiency, response time, and throughput are maximized. There are two types of scheduling algorithms used, the first type of scheduling algorithm is called preemptive, and in this type of scheduling, the processes will be interrupted based on several parameters, such as the arrival time of the process, the priority of the process, and how long the process will be executed which is called burst time. The second type of scheduling algorithm is non-preemptive, and in this type of scheduling algorithm, the processes will not be interrupted even if their parameters are different [7]. Knapsack is an optimization algorithm used to solve real-life problems involving constraints. They have two variants, namely, continuous, fractional knapsack, and discrete knapsack. The fractional knapsack is an algorithm that allows the fractional values to fill the capacity. In the case of a discrete knapsack, either element is included or excluded, and no partial values are possible [8-10]. In this research, we will use dynamic programming to solve the knapsack problem to get the maximum profit from diamonds with the appropriate weight of the shipment [8].

3. Methodology

We started by choosing a real-life problem related to customs laws. The problem was deciding the optimal shipment of diamonds based on the capacity of the knapsack the user would provide.

3.1: Dynamic Programming

Our objective is to evaluate and compare the effectiveness of the brute force and the dynamic programming approaches in resolving the Knapsack problem for optimizing diamond shipment. We collected the dataset from Kaggle, which contains information about the weight and price of diamonds. After that, we selected dynamic programming due to its optimal sub-structure, which is a pre-requisite, and brute force programming because it exhaustively checks all possible combinations and guarantees correctness. We implemented both algorithms using Python programming language. We measured the execution time in three cases, best case, average case, and worst case, for each dynamic and brute force algorithm. We also analyzed each line in both codes to find the total time and space complexity. In addition, we measured the order of growth for each of them. The results were the time and space complexity for the dynamic programming, respectively, O(n×capacity) and O(n×capacity). From the literature, it is found that the time and space complexity for the brute force respectively O(2^n) and O(n) [9].

Furthermore, we also compared dynamic programming and Genetic Algorithms to solve the 0/1 knapsack problem in terms of space and time complexity. We found that if the knapsack problem is small to medium-sized and an optimal solution is required, dynamic programming may be more efficient; for larger instances, genetic programming will be better. We made another comparison between brute force and Branch and bound to solve the 0/1 knapsack problem in terms of space and time complexity. Branch and bound are more efficient than the Brute Force algorithm. To sum up, the space complexity, best, average, and worst cases for dynamic programming are all O(n×capacity); for brute force, the time complexity is O(n), and the rest of the cases are O(2^n). Furthermore, dynamic programming is more efficient for small to medium-sized instances than genetic programming. Moreover, brute force is less efficient than Branch and bound. As a result, dynamic programming is more efficient in finding the optimal solution for the diamond shipment [10].

3.2: Genetic Algorithms

The genetic algorithm (or GA) belongs to nature-inspired, meta-heuristic, and evolutionary algorithms. It is a search method used in computing to find accurate or approximate solutions to optimization and search problems. It is beneficial when the search space is ample and the solution is approximal, like minimal or maximal. Genetic algorithms are considered to be universal search heuristic-based approximators [11-15]. GA is a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology, such as inheritance, mutation, selection, and crossover (also called recombination) [16-20]. GA is employed as a computer simulation in which an inhabitant of intangible representations (known as chromosomes or the genotype or the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem progresses about improved results. Conventionally, results are exemplified in binary as strings of 0s and 1s, but other presentations are also possible. GA's counterpart, the Differential Evolution algorithm, is utilized for the same purpose, especially for non-binary and continuous spaces [25-30]. Figure 1 shows the GA working flowchart. It starts with an initial population, usually generated randomly around an essential seed value, then fitness is evaluated, and condition to criterion is checked. If failed, the top chromosome is selected, and crossover is performed among them based on some techniques. After that, the mutation operator is applied, and this is how a new generation is generated. The process continues until results are found [31-45].

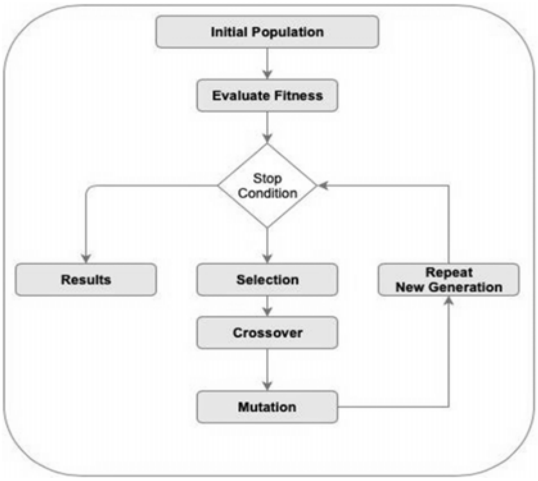

Figure 1: GA flowchart

4. Dynamic programming

As we will see, the 0-1 knapsack problem has both the optimal substructure and overlapping sub-problems needed for dynamic programming. In this 0–1 knapsack problem, we can either include or exclude a diamond from the shipment, but we cannot include it entirely or more than once. It uses a 2D table to store and reuse intermediate solutions, utilizing optimal substructure and overlapping subproblems to achieve efficiency through memoization. The retracing stage identifies specific components that contribute to the best solution. We used Python to solve this problem and analyze the time and space complexities.

4.1 Implementation

Figures 2 and 3 show the implementation of dynamic programming in Python. The program takes the knapsack's capacity, a dynamic array of items, and their weight and values, respectively. Then, based on dynamic programming principles, the items are selected optimally to fill the knapsack capacity to maximize revenue [46-50]. Moreover, in this regard, several experiments have been conducted to find the optimal value with various instances of the dataset, and several analyses have been made, as described in the subsequent sections of the article.

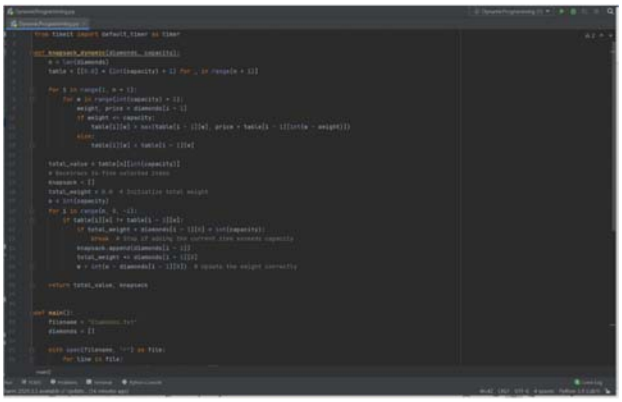

Figure 2: Dynamic programming implementation 1.

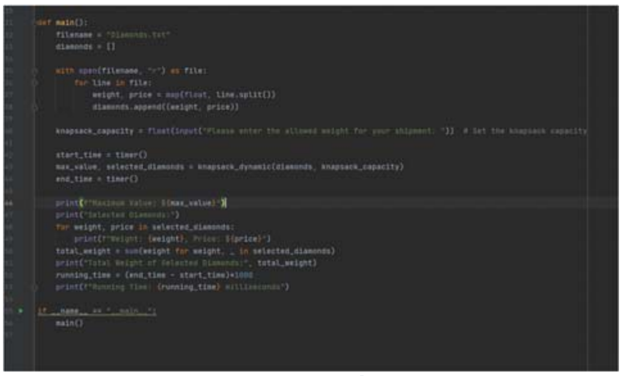

Figure 3: Dynamic programming implementation 2.

4.2 Best Case Scenario

A dedicated dataset is provided to the algorithm to observe the algorithm in terms of its best case. With that dataset, the algorithm is supposed to taper off with the minimum amount of time ideally. Here, we have a sample of the best case when we have a small dataset size (100 diamonds), and the weights and values are not large, it will have the lowest running time, measured in milliseconds, as shown in Figure 4.

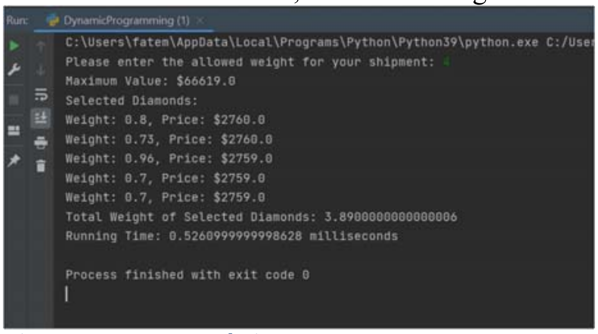

Figure 4: Best case analysis.

4.3 Average case scenario

In the average case analysis, the idea is to provide a dataset where the algorithm reaches the final solution with an average amount of time. Here, we have a sample of the average case. When we have a bigger dataset size (250 diamonds) and some large weights and values, it will take more running time than the best case. It is shown in Figure 5.

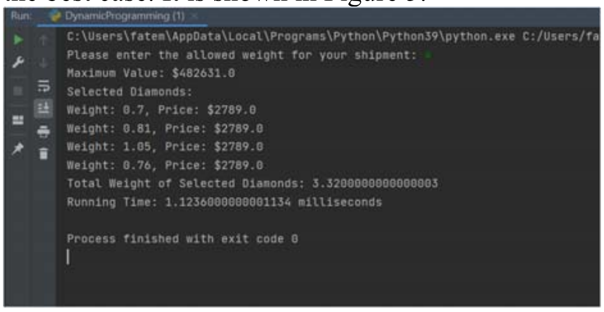

Figure 5: Average case analysis.

4.4 Worst case scenario

In the worst-case analysis, the idea is to provide a dataset where the algorithm reaches the final solution in the shortest time. Here, we have a sample of the worst case, which has a much bigger dataset size (500 diamonds) and large weight and values; it will have the longest running time. In practice, an algorithm is selected based on its worst-case running time. That algorithm performs well in the worst-case scenario and is considered the best [51-55]. This is depicted in Figure 6.

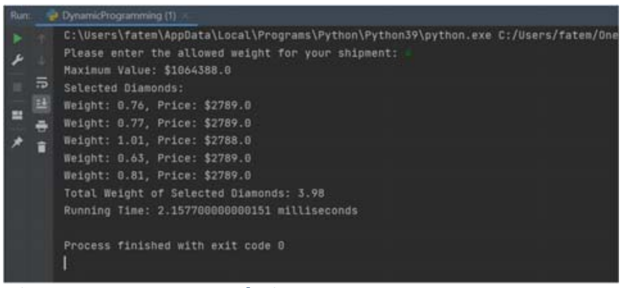

Figure 6: Worst-case analysis.

4.5 Computational complexity of Dynamic programming (analyzing line of codes):

For running time analyses, we have utilized the built-in functions of Python. The code is given below:

```
Import default_timer as timer
def knapsack_dynamic(diamonds,
capacity):
The Time and Space Complexity takes
O(1) in these two statements.
    n = len(diamonds)
    table = [[0.0] * (int(capacity) +
1) for _ in range(n + 1)]
This initializes a 2D table (table)
with dimensions (n + 1) x (capacity +
1).
The Time and Space Complexity takes
O(n*capacity).
for i in range(1, n + 1):
    for w in range(int(capacity) + 1):
The outer loop runs n times. The inner
loop runs int(capacity) + one time.
The Time Complexity takes O(n*capacity)
and the Space Complexity O(1)
        weight, price = diamonds[i - 1]
        if weight <= capacity:
           table[i][w] = max(table[i -
1][w], price + table[i - 1][int(w -
weight)])
        else:
          table[i][w] = table[i - 1][w]
Each line's time and space complexity
inside the loop is constant (1). This
loop iterates through 'n' components
and has an inner loop of 'capacity'
iterations; as a result, its overall
complexity is O(n * capacity) in time
and O(n * capacity) in space.
total_value = table[n][int(capacity)]
# Backtrack to find selected items
knapsack = []
total_weight = 0.0  # Initialize total
weight
```

```
w = int(capacity)
The Time and Space Complexity takes
O(1).
for i in range(n, 0, -1):
    if table[i][w] != table[i - 1][w]:
        if total_weight + diamonds[i -
1][0] > int(capacity):
            break  # Stop if adding the
current item exceeds the capacity
       knapsack.append(diamonds[i - 1])
     total_weight += diamonds[i - 1][0]
        w = int(w - diamonds[i - 1][0])
# Update the weight correctly
Time Complexity: O(n) iterates through
'n' elements.
Space Complexity: O(1) - Constant space
for the loop variables.
return total_value, knapsack
The Time and Space Complexity takes
O(1).
filename = "Diamonds.txt"
diamonds = []
with open(filename, "r") as file:
    for line in file:
        weight, price = map(float,
line.split())
       diamonds.append((weight, price))
Time Complexity: O(n) - It iterates
through the 'n' lines in the file.
Space Complexity: O(n) - It stores 'n'
tuples in the 'diamonds' list.
knapsack_capacity = float(input("Please
enter the allowed weight for your
shipment: "))  # Set the knapsack
capacity
The Time and Space Complexity takes
O(1).
start_time = timer()
max_value, selected_diamonds =
knapsack_dynamic(diamonds,
knapsack_capacity)
end_time = timer()
Time Complexity: O(n * capacity) - It
calls the 'knapsack_dynamic' function.
Space Complexity: O(n * capacity) - It
depends on the space complexity of the
'knapsack_dynamic' function.
print(f"Maximum Value: ${max_value}")
print("Selected Diamonds:")
for weight, price in selected_diamonds:
    print(f"Weight: {weight}, Price:
${price}")
Time Complexity: O(n) - It iterates
through 'n' elements in
'selected_diamonds.'
Space Complexity: O(1) - Constant space
for printing.
total_weight = sum(weight for weight, _
in selected_diamonds)
print("Total Weight of Selected
Diamonds:", total_weight)
Time Complexity: O(n) - It iterates
through 'n' elements in
'selected_diamonds' by the sum
operation.
Space Complexity: O(1) - Constant space
for the 'total_weight' variable.
running_time = (end_time -
start_time)*1000
print(f"Running Time: {running_time}
milliseconds")
Time and Space Complexity takes O(1).
```

4.6. Time complexity T(n) & Order of growth for dynamic programming

After doing the code analysis, we have concluded that the overall time complexity T(n) and order of growth is O(n×capacity). Depending on the knapsack's capacity, the order of growth regarding the dataset size is linear in n. Considering n, the time complexity is linear if the capacity is constant. Nevertheless, the time complexity is O(n×capacity) if the capacity is variable and can increase with the collection amount. Figure 7 shows the order of growth for Dynamic programming.

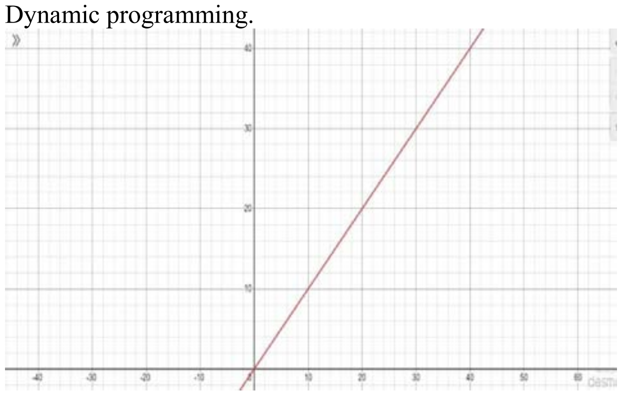

Figure 7: Order of growth For Dynamic programming

4.8 Comparison between dynamic programming and genetic algorithm

This section compares dynamic programming and genetic algorithms for solving the knapsack problem. Table 1 provides a comparison.

5. Brute Force Algorithm

The second algorithm that solves a Knapsack problem is the brute force algorithm. Which will find the best combinations by trying all the inputs; it will accept the combination if it produces a total weight for the shipment less than or equal to the shipment capacity. Note that the 0/1 knapsack algorithm takes or rejects the whole item; no fractions are allowed! [56] For the data collected, we used a Diamond dataset from the Kaggle website and Python programming language for coding.

Table 1: Comparison between Dynamic Programming and Genetic Algorithms

Comparison Feature	Dynamic programming	Genetic Algorithm
Approaches	Constructs a table with each cell representing the maximum value that can be reached with a specific weight and a subset of the items. The value of including or removing each item for each weight is then compared to fill the table [57].	It solves the 0/1 knapsack problem. Using selection, crossover, and mutation, GA generates a broad range of viable solutions that gradually focus on the best option. Its performance depends on parameter settings like population size and mutation rate, which require fine-tuning for optimal results [58].
Time complexity	O(N*W)	O (n*population (size*generation size)
Space complexity	O(N*W)	O (n*population size*generation size)
Efficiency	Dynamic programming may be more efficient if the knapsack problem is small to medium-sized and an optimal solution is required. Genetic algorithms may perform better for more significant	instances or when an approximate solution is acceptable.

5.1 Time complexity T(n) & Order of growth For Brute Force Algorithm

The time complexity is exponential (2^n), where n is the size of the dataset. In the case of brute force, it generates all the possible solutions. The order of growth is also exponential because we have nested loops that iterate through all the combinations of diamonds. In the inner loop, we create the combinations using itertools.combinations.
In the outer loop, it iterates several times, equal to the number of diamonds; that is why it is exponential in nature. Its behavior is depicted in Figure 8 where it shoots up even with fewer iterations on x-axis.

Running time

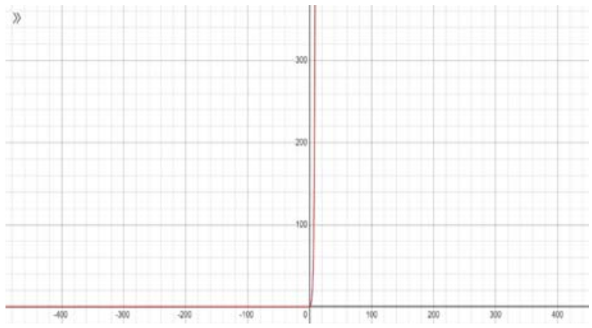

Figure 8: Growth rate of Brute Force algorithm

5.2 Comparison between brute force and branch-and-bound.

Table 2 presents a comparison between brute force and branch-and-bound.

Table 2: Brute force and Branch & bound comparison

Table 3 compares Brute force and Dynamic Programming algorithms based on three cases.

Table 3: Comparison between Dynamic Algorithm and Brute Force Cases

Algorithm	Dynamic programming	Brute force
Space complexity	O(nW)	O(n)
Best	O(nW)	O($\boldsymbol{2^n}$)
Worst	O(nW)	O($\boldsymbol{2^n}$)
Average	O(nW)	O($\boldsymbol{2^n}$)

Similarly, Table 4 compares dynamic programming and brute force regarding execution time.

Table 4: Comparison between Dynamic Algorithm & Brute Force Algorithm based on T(n)

#	Input size	Dynamic Programming O(N.W)	Brute Force Algorithm $\boldsymbol{O}(\boldsymbol{2^n})$
1.	10	0.09883599999982451 ms	5.465989000000171 ms
2.		0.09731500000009774 ms	7.157218000000021 ms
3.		0.07336699999993535 ms	6.826498999999986 ms
4.		0.10491800000000939 ms	6.023648999999964 ms
5.		0.09617500000014267 ms	5.998559999999986 ms
6.		0.07488699999991244 ms	5.162638999999913 ms
7.		0.10149600000008974 ms	5.239427999999879 ms
8.		0.07526699999993447 ms	4.965348000000036 ms
9.		0.07146499999999278 ms	4.902626000000021 ms
10.		0.06766400000002282 ms	5.513886000000134 ms
Average running time		0.0961308 ms	5.725544 ms
1.	20	0.11632199999977999 ms	7420.097306999999 ms
2.		0.1672600000000024 ms	7249.251606000001 ms
3.		0.11670199999991304 ms	7485.677390999999 ms
4.		0.11404100000000916 ms	7225.299178 ms
5.		0.1075790000000243 ms	7720.121142 ms
6.		0.18550699999986264 ms	7356.522067000001 ms
7.		0.11860299999999491 ms	7454.425952999999 ms
8.		0.1170820000000461 ms	7851.035739999999 ms
9.		0.14635299999987694 ms	7045.72249 ms
10.		0.1174619999999571 ms	7767.069193000001 ms
Average running time		0.1207911 ms	7377.522107 ms

After comparing, it was found that the dynamic programming works more efficiently in terms of time. This benefit is ascribed to its capacity to use overlapping subproblems and optimal substructure, which minimizes computation duplication. Dynamic programming is more scalable for more significant problem instances because it achieves a more favorable time complexity. Nevertheless, because subproblem solutions must be stored, it comes at the expense of more space complexity. Depending on the nature of the task and the resources at hand, one can choose between these approaches; in general, dynamic programming provides a more effective solution.

Comparison Feature	**Brute Force Algorithm**	**Branch and bound**
Approaches	The Brute force algorithm explores all possible solutions and finds the best solution. In the 0/1 knapsack problem, the brute force algorithm tries to find the maximum value with a weight less than or equal to the bag size [59].	An effective way to solve the 0/1 knapsack problem is to pay attention to unhelpful solutions. For example, we can ignore a node and its subtrees if the best in the subtree is worse than the current best. Therefore, before exploring a node, we first calculate each node's bound (best solution) and compare it with the current best solution [60].
Time complexity	O(2^n)	O(2^n)
Space complexity	O(n)	O(2^n)
Efficiency	Less efficient.	More efficient.

6. Conclusion

To sum up, to identify which approach would result in the most workable and effective solution for the problem, we investigated the Knapsack Problem. We carefully examined and compared four different methods for solving it: dynamic programming, genetic algorithm, brute force, and Branch and bound methods. Dynamic programming is widely known for its remarkable effectiveness in breaking down complicated problems into smaller subproblems that may be solved. Because of its significantly lower space and time-based complexity than the other methods, we found that it is the optimal solution for more complex versions of the Knapsack problem. On the other hand, although the Brute Force Method ensured the best possible solution, its exponential temporal complexity caused inherent problems. Because of its exhaustive search method for issue solving, it ensured correctness while being impractical for larger datasets due to its comprehensive study of every possible combination. As far as the genetic algorithm is concerned, it is best suited to situations where the search space is considerably large and the solution is to be found heuristically. In the future, we intend to investigate non-deterministic polynomial (NP) and NP-hard problems using more heuristic-based and hybrid algorithms [61-65].

References

[1]. Terh, F. (2019). How to solve the Knapsack problem with dynamic programming. Retrieved from https://medium.com/@fabianterh/how-to-solve-the-knapsack-problem-with-dynamic-programming-eb88c706d3cf

[2]. Al-Fareed, H., Alghamdi, O., Alshuraya, A., Alqahtani, M., Alwasfer, S., Aljomea, A., Rahman, A., Aljameel, S., Krishnasamy, G. (2022). Simulator for scheduling real-time systems with reduced power consumption. Mathematical Modelling of Engineering Problems, Vol. 9, No. 5, pp. 1225-1232.

[3]. W. Hantom, A. Aldweesh, R. Alzaher, A. Rahman, "A Survey on Scheduling Algorithms in Real-Time Systems," IJCSNS - International Journal of Computer Science and Network Security 22(4), 686-690, 2022.

[4]. N. AlDossary, S. AlQahtani, H. AlUbaidan, A. Rahman, "A Survey on Resource Allocation Algorithms and Models in Cloud Computing," IJCSNS International Journal of Computer Science and Network Security 22 (3), 776-782, 2022.

[5]. A. Obregon, "Introduction to sorting algorithms in java: A beginner's guide," Medium, https://medium.com/@AlexanderObregon/introduction-to-sorting-algorithms-in-java-abeginners-guide-db522047effb (accessed Nov. 30, 2023).

[6]. "Counting sort - data structures and algorithms tutorials," GeeksforGeeks, https://www.geeksforgeeks.org/counting-sort/ (accessed Nov. 30, 2023).

[7]. I. Qureshi, "CPU Scheduling Algorithms: A Survey," Int. J. Advanced Networking and Applications, vol. 5, no. 4, pp. 1968-2973, 2014.

[8]. I. Alrashide, H. Alkhalifah, A.A. Al-Momen, I. Alali, G. Alshaikh et al., "AIMS: AI based Mental Healthcare System," IJCSNS - International Journal of Computer Science and Network Security 23(12), 225-234, 2023.

[9]. A. Alhashem, A. Abdulbaset, F. Almudarra, H. Alshareef, M. Alqasoumi et al., "Diabetes Detection and Forecasting using Machine Learning Approaches: Current State-of-the-art," IJCSNS - International Journal of Computer Science and Network Security 23(10), 199-208, 2023.

[10]. A. Albassam, F. Almutairi, N. Majoun, R. Althukair, Z. Alturaiki et al., "Integration of Blockchain and Cloud Computing in Telemedicine and Healthcare," IJCSNS - International Journal of Computer Science and Network Security 23(6), 17-26, 2023.

[11]. M Mahmud, A Rahman, M Lee, JY Choi, "Evolutionary-based image encryption using RNA codons truth table," Optics & Laser Technology 121, 105818, 2020.

[12]. Atta-ur-Rahman, Dash, S., Luhach, A.K. et al. A Neuro-fuzzy approach for user behaviour classification and prediction. J Cloud Comp 8, 17 (2019). https://doi.org/10.1186/s13677-019-0144-9.

[13]. Atta-ur-Rahman, Sultan, K., Aldhafferi, N., Alqahtani, A. (2018). Differential Evolution Assisted MUD for MC-CDMA Systems Using Non-Orthogonal Spreading Codes. In: Abraham, A., Haqiq, A., Muda, A., Gandhi, N. (eds) Innovations in Bio-Inspired Computing and Applications. IBICA 2017. Advances in Intelligent Systems and Computing, vol 735. Springer, Cham.

[14]. A Rahman, M Mahmud, K Sultan, N Aldhafferi, A Alqahtani, D Musleh, "Medical Image Watermarking for Fragility and Robustness: A Chaos, ECC and RRNS Based Approach," Journal of Medical Imaging and Health Informatics 8 (6), 1192-1200, 2018.

[15]. A Rahman, IM Qureshi, AN Malik, MT Naseem, "Dynamic resource allocation in OFDM systems using DE and FRBS," Journal of Intelligent and Fuzzy Systems 26 (4), 2035-2046, 2014.

[16]. Atta-ur-Rahman, D. -e. -N. Zaidi, M. H. Salam and S. Jamil, "User behaviour classification using Fuzzy Rule Based System," 13th International Conference on Hybrid Intelligent Systems (HIS 2013), Gammarth, Tunisia, 2013, pp. 117-122.

[17]. Atta-Ur-Rahman, I. M. Qureshi, M. H. Salam and M. Z. Muzaffar, "Adaptive communication using softcomputing techniques," 2013 International Conference on Soft Computing and Pattern Recognition (SoCPaR), Hanoi, Vietnam, 2013, pp. 19-24, doi: 10.1109/SOCPAR.2013.7054131.

[18]. Atta-ur-Rahman, M. H. Salam, M. T. Naseem and M. Z. Muzaffar, "An intelligent link adaptation scheme for OFDM based Hyperlans," 2013 International Conference on Soft Computing and Pattern Recognition (SoCPaR), Hanoi, Vietnam, 2013, pp. 360-365, doi: 10.1109/SOCPAR.2013.7054159.

[19]. Atta-ur-Rahman, I. M. Qureshi, M. H. Salam and M. T. Naseem, "Efficient link adaptation in OFDM systems using a hybrid intelligent technique," 13th International Conference on Hybrid Intelligent Systems (HIS 2013), Gammarth, Tunisia, 2013, pp. 12-17, doi: 10.1109/HIS.2013.6920471.

[20]. RA Qamar, M Sarfraz, A Rahman, SA Ghauri, "Multi-Criterion Multi-UAV Task Allocation under Dynamic Conditions," Journal of King Saud University-Computer and Information Sciences 35 (9), 101734, 2023.

[21]. Z Alsadeq, H Alubaidan, A Aldweesh, A Rahman, T Iqbal, "A Proposed Model for Supply Chain using Blockchain Framework," IJCSNS - International Journal of Computer Science and Network Security 23(6), 91-98, 2023.

[22]. S. Arooj, M. F. Khan, T. Shahzad, M. A. Khan, M. U. Nasir et al., "Data fusion architecture empowered with deep learning for breast cancer classification," Computers, Materials & Continua, vol. 77, no.3, pp. 2813–2831, 2023.

[23]. Jan, F.; Rahman, A.; Busaleh, R.; Alwarthan, H.; Aljaser, S.; Al-Towailib, S.; Alshammari, S.; Alhindi, K.R.; Almogbil, A.; Bubshait, D.A.; et al. Assessing Acetabular Index Angle in Infants: A Deep Learning-Based Novel Approach. J. Imaging 2023, 9, 242.

[24]. M. M. Qureshi, F. B. Yunus, J. Li, A. Ur-Rahman, T. Mahmood and Y. A. A. Ali, "Future Prospects and Challenges of On-Demand Mobility Management Solutions," in IEEE Access, vol. 11, pp. 114864-114879, 2023, doi: 10.1109/ACCESS.2023.3324297.

[25]. M. Gollapalli, A. -U. Rahman, A. Osama, A. Alfaify, M. Yassin and A. Alabdullah, "Data Mining and Visualization to Understand Employee Attrition and Work Performance," 2023 3rd International Conference on Computing and Information Technology (ICCIT), Tabuk, Saudi Arabia, 2023, pp. 149-154, doi: 10.1109/ICCIT58132.2023.10273889.

[26]. Musleh, D.A.; Olatunji, S.O.; Almajed, A.A.; Alghamdi, A.S.; Alamoudi, B.K.; Almousa, F.S.; Aleid, R.A.; Alamoudi, S.K.; Jan, F.; Al-Mofeez, K.A.; et al. Ensemble Learning Based Sustainable Approach to Carbonate Reservoirs Permeability Prediction. Sustainability 2023, 15, 14403.

[27]. Ahmed, M.I.B.; Saraireh, L.; Rahman, A.; Al-Qarawi, S.; Mhran, A.; Al-Jalaoud, J.; Al-Mudaifer, D.; Al-Haidar, F.; AlKhulaifi, D.; Youldash, M.; et al. Personal Protective Equipment Detection: A Deep-Learning-Based Sustainable Approach. Sustainability 2023, 15, 13990.

[28]. Ahmed, M.I.B.; Alabdulkarem, H.; Alomair, F.; Aldossary, D.; Alahmari, M.; Alhumaidan, M.; Alrassan, S.; Rahman, A.; Youldash, M.; Zaman, G. A Deep-Learning Approach to Driver Drowsiness Detection. Safety 2023, 9, 65.

[29]. Ahmed, M.I.B.; Alotaibi, R.B.; Al-Qahtani, R.A.; Al-Qahtani, R.S.; Al-Hetela, S.S.; Al-Matar, K.A.; Al-Saqer, N.K.; Rahman, A.; Saraireh, L.; Youldash, M.; et al. Deep Learning Approach to Recyclable Products Classification: Towards Sustainable Waste Management. Sustainability 2023, 15, 11138.

[30]. Sajid, N.A.; Rahman, A.; Ahmad, M.; Musleh, D.; Basheer Ahmed, M.I.; Alassaf, R.; Chabani, S.; Ahmed, M.S.; Salam, A.A.; AlKhulaifi, D. Single vs. Multi-Label: The Issues, Challenges and Insights of Contemporary Classification Schemes. Appl. Sci. 2023, 13, 6804.

[31]. Gollapalli, M.; Rahman, A.; Alkharraa, M.; Saraireh, L.; AlKhulaifi, D.; Salam, A.A.; Krishnasamy, G.; Alam Khan, M.A.; Farooqui, M.; Mahmud, M.; et al. SUNFIT: A Machine Learning-Based Sustainable University Field Training Framework for Higher Education. Sustainability 2023, 15, 8057.

[32]. Talha, M.; Sarfraz, M.; Rahman, A.; Ghauri, S.A.; Mohammad, R.M.; Krishnasamy, G.; Alkharraa, M. Voting-Based Deep Convolutional Neural Networks (VB-DCNNs) for M-QAM and M-PSK Signals Classification. Electronics 2023, 12, 1913.

[33]. T. A. Khan et al., "Secure IoMT for Disease Prediction Empowered with Transfer Learning in Healthcare 5.0, the Concept and Case Study," in IEEE Access, vol. 11, pp. 39418-39430, 2023, doi: 10.1109/ACCESS.2023.3266156.

[34]. Musleh, D.; Alotaibi, M.; Alhaidari, F.; Rahman, A.; Mohammad, R.M. Intrusion Detection System Using Feature Extraction with Machine Learning Algorithms in IoT. J. Sens. Actuator Netw. 2023, 12, 29. https://doi.org/10.3390/jsan12020029.

[35]. Alghamdi, A.S.; Rahman, A. Data Mining Approach to Predict Success of Secondary School Students: A Saudi Arabian Case Study. Educ. Sci. 2023, 13, 293.

[36]. MA Qureshi, M Asif, S Anwar, U Shaukat, MA Khan, A Mosavi, "Aspect Level Songs Rating Based Upon Reviews in English," Computers, Materials & Continua 74 (2), 2589-2605, 2023.

[37]. NA Sajid, M Ahmad, A Rahman, G Zaman, MS Ahmed, N Ibrahim et al., "A Novel Metadata Based Multi-Label Document Classification Technique," Computer Systems Science and Engineering 46 (2), 2195-2214, 2023.

[38]. Basheer Ahmed, M.I.; Zaghdoud, R.; Ahmed, M.S.; Sendi, R.; Alsharif, S.; Alabdulkarim, J.; Albin Saad, B.A.; Alsabt, R.; Rahman, A.; Krishnasamy, G. A Real-Time Computer Vision Based Approach to Detection and Classification of Traffic Incidents. Big Data Cogn. Comput. 2023, 7, 22.

[39]. Alqarni, A.; Rahman, A. Arabic Tweets-Based Sentiment Analysis to Investigate the Impact of COVID-19 in KSA: A Deep Learning Approach. Big Data Cogn. Comput. 2023, 7, 16.

[40]. S Abbas, SA Raza, MA Khan, A Rahman, K Sultan, A Mosavi, "Automated File Labeling for Heterogeneous Files Organization Using Machine Learning," Computers, Materials & Continua 74 (2), 3263-3278, 2023.

[41]. MS Farooq, S Abbas, A Rahman, K Sultan, MA Khan, A Mosavi, "A Fused Machine Learning Approach for Intrusion Detection System," Computers, Materials & Continua 74 (2), 2607–2623, 2023.

[42]. Alhaidari, F., Rahman, A. & Zagrouba, R. Cloud of Things: architecture, applications and challenges. J Ambient Intell Human Comput 14, 5957–5975 (2023). https://doi.org/10.1007/s12652-020-02448-3.

[43]. Rahman, A. GRBF-NN based ambient aware realtime adaptive communication in DVB-S2. J Ambient Intell Human Comput 14, 5929–5939 (2023).

[44]. Ahmad, M., Qadir, M.A., Rahman, A. et al. Enhanced query processing over semantic cache for cloud-based relational databases. J Ambient Intell Human Comput 14, 5853–5871 (2023).

[45]. M Jamal, NA Zafar, D Musleh, MA Gollapalli, S Chabani, "Modeling and Verification of Aircraft Takeoff Through

Novel Quantum Nets," Computers, Materials & Continua 72 (2), 3331-3348, 2022.
[46]. M.U. Nasir, T.M. Ghazal, M.A. Khan, M. Zubair, Atta-ur Rahman, R. Ahmed, H. AlHamadi, C.Y.Yeun, "Breast Cancer Prediction Empowered with Fine-Tuning", Computational Intelligence and Neuroscience, vol. 2022, Article ID 5918686, 2022.
[47]. A Rahman, M Ahmed, G Zaman, T Iqbal, MAA Khan et al., "Geo-Spatial Disease Clustering for Public Health Decision Making," Informatica 46 (6), 21-32, 2022.
[48]. Atta-ur-Rahman, Ibrahim, N.M., Musleh, D., Khan, M.A.A., Chabani, S., Dash, S. (2022). Cloud-Based Smart Grids: Opportunities and Challenges. In: Dehuri, S., Prasad Mishra, B.S., Mallick, P.K., Cho, SB. (eds) Biologically Inspired Techniques in Many Criteria Decision Making. Smart Innovation, Systems and Technologies, vol 271. Springer, Singapore.
[49]. M.B.S Khan, Atta-ur-Rahman, M.S. Nawaz, R. Ahmed, M.A. Khan, A. Mosavi. Intelligent breast cancer diagnostic system empowered by deep extreme gradient descent optimization[J]. Mathematical Biosciences and Engineering, 2022, 19(8): 7978-8002. doi: 10.3934/mbe.2022373.
[50]. F Al-Jawad, R Alessa, S Alhammad, B Ali, M Al-Qanbar, A Rahman, "Applications of 5G and 6G in Smart Health Services," IJCSNS, 22 (3): 173-182, 2022.
[51]. A Rahman, K Sultan, I Naseer, R Majeed, D Musleh et al., "Supervised machine learning-based prediction of COVID-19," Computers, Materials and Continua 69 (1), 21-34, 2021.
[52]. R Zagrouba, A AlAbdullatif, K AlAjaji et al., "Authenblue: A New Authentication Protocol for the Industrial Internet of Things," Computers, Materials & Continua 67 (1), 1103-1119, 2021.
[53]. G Zaman, H Mahdin, K Hussain, A Rahman, et al., "Digital Library of Online PDF Sources: An ETL Approach," IJCSNS 20 (11), 172-181, 2020.
[54]. A Rahman, "Memetic computing based numerical solution to Troesch problem," Journal of Intelligent and Fuzzy Systems 36 (6), 1-10, 2019.
[55]. A Rahman, "Optimum information embedding in digital watermarking," Journal of Intelligent and Fuzzy Systems 37 (1), 553-564, 2019.
[56]. Pan, X. and Zhang, T. (2018) 'Comparison and analysis of algorithms for the 0/1 Knapsack problem', Journal of Physics: Conference Series, 1069, p. 012024. doi:10.1088/1742-6596/1069/1/012024.
[57]. Algorithm Design, "What are the pros and cons of dynamic programming vs. greedy methods for the knapsack problem?" *www.linkedin.com*. Accessed on Nov. 20, 2023.
[58]. T. Pradhan, A. Israni and M. Sharma, "Solving the 0–1 Knapsack problem using Genetic Algorithm and Rough Set Theory," IEEE International Conference on Advanced Communications, Control and Computing Technologies, Ramanathapuram, India, 2014, pp. 1120-1125.
[59]. Florian, "0/1 Knapsack problem," Medium, https://medium.com/@florian_algo/0-1-knapsack-problem-eec333f4a991 (accessed Nov. 30, 2023).
[60]. "0/1 knapsack using branch and bound," GeeksforGeeks, https://www.geeksforgeeks.org/0-1-knapsack-using-branch-and-bound/ (accessed Nov. 30, 2023).
[61]. I.A. Qureshi, K.A. Bhatti, A. Rahman, et al., "GFuCWO: A genetic fuzzy logic technique to optimize contention window of IEEE-802.15. 6 WBAN," Ain Shams Engineering Journal, 10268, 2024, doi: 10.1016/j.asej.2024.102681.
[62]. M.A. Khan, S. Abbas, A. Atta et al., "Intelligent cloud based heart disease prediction system empowered with supervised machine learning," Computers, Materials & Continua 65 (1), 139-151, 2020.
[63]. A Rahman, IM Qureshi, AN Malik, "Adaptive Resource Allocation in OFDM Systems Using GA and Fuzzy Rule Base System," World Applied Sciences Journal 18 (6), 836-844, 2021.
[64]. S Dash, BK Tripathy, A Rahman, "Handbook of Research on Modeling, Analysis, and Application of Nature-Inspired Metaheuristic Algorithms," IGI Global 1, 540, 2017.
[65]. M. T. Naseem, I. M. Qureshi, A. Rahman and M. Z. Muzaffar, "Robust watermarking for medical images resistant to geometric attacks," in Proc. INMIC, Islamabad, Pakistan, 2012, pp. 224-228.