Kasu Journal of Computer Science Vol. 1 No. 2 [June, 2024], pp. 412-422
https://doi.org//10.47514/kjcs/2024.1.2.0017

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

Exploring Heuristic Algorithms for the Knapsack Problem: A Comparative Analysis of Program Complexity and Computational Efficiency

Bashar Bin Usman[1], Saminu Isah Kanoma[2], Ibrahim Zakariyau [3] and Hashimu Abubakar Niworu [4] and Adebayo Ademola Rilwan[5].

[1]Department of Computer Science, Faculty of Natural Sciences, Ibrahim Badamasi Babangida University, Lapai, Nigeria.

[2]ICT/MIS Directorate, Federal University Gusau, Nigeria

[3]Department of information and Documentation, Oceanography and Marine Research, Lagos, Nigeria

[4]Department of Computer Science, Federal College of Wildlife Management, New Bussa, Nigeria

[5]Computer Science, Federal School of Surveying, Oyo P.M.B 1024

basharbinusman1@gmail.com[1], sikanoma4u@gmail.com[2], zakariibrahim@niomr.gov.ng[3], hashimuniworu2525@gmal.com[4] and aadebayo22@yahoo.com[5]

Abstract

The knapsack problem is an optimization problem in computer science which involves determining the most valuable combination of items that can be packed into a knapsack (a container) with a limited capacity (weight or volume); the goal is to maximize the total profit of the items included in the knapsack without exceeding its capacity. This study extensively analyzes the knapsack problem, exploring the application of three prevalent heuristics: greedy, dynamic programming, and FPTAS algorithms implemented in Python. The study aims to assess how these algorithms perform differently, focusing on program complexity and computational speed. Our main objective is to compare these algorithms and determine the most effective one for solving the knapsack problem as well as to be chosen by the researchers and developers when dealing similar problem in real-world applications. Our methodology involved solving the knapsack problem using the three algorithms within a unified programming environment. We conducted experiments using varying input datasets and recorded the time complexities of the algorithms in each trial. Additionally, we performed Halstead complexity measurements to derive the volume of each algorithm for this study. Subsequently, we compared program complexity in Halstead metrics and computational speed for the three approaches. The research findings reveal that the Greedy algorithm demonstrates superior computational efficiency compared to both Dynamic Programming (D.P) and FPTAS algorithms across various test cases. To advance understanding of the knapsack problem, future research should focus on investigating the performance of other programming languages in addressing combinatorial optimization problems, which would provide valuable insights into language choice impact. Additionally, integrating parallel computing techniques could accelerate solution processes for large-scale problem instances.

Keywords: Knapsack problem, Heuristics, Computational efficiency, Halstead metrics Python programming, Combinatorial optimization

Kasu Journal of Computer Science Vol. 1 No. 2 [June, 2024], pp. 412-422
https://doi.org//10.47514/kjcs/2024.1.2.0017

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

1. Introduction

The knapsack problem is an optimization problem in computer science, commonly applied in various fields, which involves finding the most valuable combination of items that can be packed into a container with a limited capacity, whether by weight or volume. Specifically, it involves determining the optimal selection of items (n), each with a weight (w_i) and a profit (p_i), to maximize the total value while keeping the total weight within a given limit (M) (the capacity of the knapsack) (Kellerer et al.,2004).

$$\text{Maximize} \sum_{1\le i\le n} p_i x_i \qquad (1)$$

$$\text{subject to} \sum_{1\le i\le n} w_i x_i \le M \qquad (2)$$

$$\text{and } 0<x_i<1,\ p_i>0,\ w_i>0,\ 1<i<n, \qquad (3)$$

A feasible solution or filling is any set (x_1, x_2,.....x_n), satisfying equations (2) and (3), while an optimal solution is a feasible solution for which (1) is maximum. (Kellerer et al. 2004). Two versions of the knapsack problem are the 0-1 knapsack problem and the fractional knapsack problem (Goddard, 2015).To solve these problems for optimal or near-optimal solutions, various heuristics have been developed: This study leverage the greedy, dynamic programming, and FPTAS algorithms to solving the knapsack problem. Furthermore, this study aims to explore how these algorithms perform differently, focusing on their program complexity and computational speed. Our main focus is to compare these algorithms and determine which one is most effective for solving the knapsack problem as well as to be chosen by the programmers when dealing similar problem in real-world applications such as logistics, telecommunications, internet data packet routing, manufacturing, finance, and healthcare. This paper is structured as follows: Section 2 provides an overview of related works. In Section 3, we present the conceptual framework, including discussions on Greedy, Dynamic Programming, and the FPTAS algorithms. Section 4 presents the Experimental results and discuss the research findings while Section 6 outlines the Conclusion.

2. Related Works

Several studies have compared dynamic programming and greedy algorithms for solving the knapsack problem: San Lin Aung (2019) conducted a comparative study between Dynamic Programming and the Greedy Algorithm to solve the Knapsack Problem, which aims to maximize the benefit of items in a knapsack without exceeding its capacity. The results of this study demonstrated that the Dynamic Programming algorithm outperformed the Greedy Algorithm in terms of the time taken and the total value generated. While the Greedy Algorithm lacks parallelism, the Dynamic Programming Algorithm possesses this property. However, the scope of this research was limited to maximizing knapsack capacity and time

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

only. Namer Ali Al Etawi and Fatima Thaher Aburomman (2020) conducted a comparative study between Greedy and Dynamic Programming algorithms for solving the Knapsack problem. Their research demonstrated that the Greedy algorithm outperforms Dynamic Programming in terms of runtime efficiency. However, this study is limited by the fact that item weights were randomly generated using the JavaRandom.next() method. Xiaoxi Chen's (2022) study explores greedy and dynamic programming algorithms for the knapsack problem, emphasizing time complexity. The research highlights the efficiency of the greedy algorithm, providing faster results, though not always optimal. In contrast, dynamic programming ensures optimal solutions but with slower computation. The comparison underscores the superior time complexity of the greedy algorithm in knapsack problem-solving, focusing on these two algorithms exclusively. Yitan Wu (2022), analyzed strategies for the 0-1 knapsack problem within real-life cargo delivery scenarios. Comparing Dynamic Programming and Greedy algorithms, the study aimed to aid decision-making in practical situations. Recommending Greedy for large-scale problems due to time efficiency and Dynamic Programming for precision in small-scale scenarios, the research acknowledges the limitations of both methods. While insightful, the research could be improved by looking into extra methods. Bin Usman et al., (2023), study the efficiency of three algorithms greedy, dynamic programming, and branch-and-bound was compared. C++ implementations revealed that the greedy approach was the most efficient heuristic, followed closely by dynamic programming. In contrast, the branch-and-bound algorithm exhibited the least efficiency among the three. While all the above studies provide valuable insights into algorithmic efficiencies for the knapsack problem, there is a potential research gap in exploring more advanced or novel algorithms beyond Dynamic Programming, Greedy, and branch-and-bound. Specifically, there is an opportunity to investigate the effectiveness of other advanced algorithms such as metaheuristics like FPTAS (Fully Polynomial Time Approximation Scheme) in different programming language.

3. Methodology

This research extensively delves into the complexities of the knapsack problem, with a specific emphasis on the exploration of three methods: the greedy, dynamic programming, and FPTAS algorithms for addressing the knapsack problem. We compared these algorithms and evaluated their efficiency in solving the knapsack problem. To achieve this, we conducted experimental runs and analyzed the results. The overarching purpose of this study is to understand the strengths and weaknesses of these strategies for solving knapsack problems, such as resource allocation, financial portfolio management, and logistics. By considering these algorithms program volume and time complexities, we provide a thorough assessment of algorithmic performance, offering recommendations for theorists and programmers facing similar challenges. The research focus is depicted graphically in figure 3.1.

Kasu Journal of Computer Science Vol. 1 No. 2 [June, 2024], pp. 412-422
https://doi.org//10.47514/kjcs/2024.1.2.0017

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

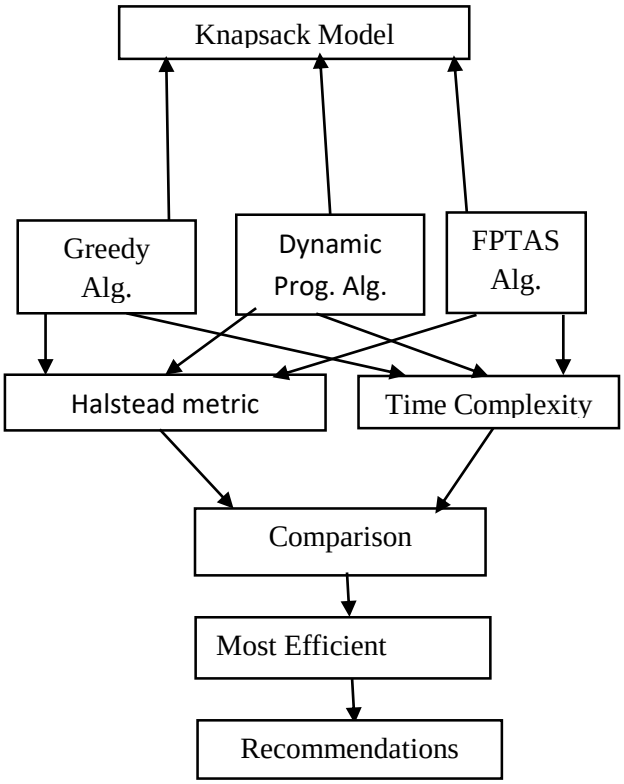


Figure 3.1: Research Conceptual Diagram (Source: Bashar etal., 2024)

3.1 Implementation of Strategies for solving the Knapsack problem:

a. Implementation of the Greedy Approach:

The greedy algorithm for the knapsack problem involves selecting items based on a greedy criterion, prioritizing those with the highest value-to-weight ratio. It iteratively adds items to the knapsack until the weight limit is reached.

```
Greedy (p,w,c,i)

//objective: To obtain the maximum profit of the knapsack

// input: list of items, each with a profit pi and a weight wi

// the capacity of knapsack c

// output: the maximum profit made by filling the knapsack

    for i=1 to n do

        x=select (w)

     if  feasible (x) then

        solution= solution+x
```

Kasu Journal of Computer Science Vol. 1 No. 2 [June, 2024], pp. 412-422
https://doi.org//10.47514/kjcs/2024.1.2.0017
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

Endif

Repeat

Return (solution)

b. Implementation of the Dynamic Programming Approach:

The dynamic programming approach is an optimization method that efficiently addresses the knapsack problem by decomposing it into smaller subproblems. It establishes a table to store the maximum value achievable at each capacity, considering solutions from previous subproblems. Through iterative population of the table, it guarantees optimal solutions for each subproblem, ultimately culminating in the identification of the overall maximum value.

```
Dynamicalg(p, w, c, i, j, n, t)

// Objective: To obtain the maximum profit of the knapsack

// Input: list of items, each with a profit pi and a weight wi, The capacity of knapsack c

// Output: the maximum profit made by filling the knapsack

For (int i=0; i<=n, i++)

{
   For (int j=0; j<=c, j++)
   {
      If (i=0, && j=o)
         t[i,j]=0;
      Elseif
         (wi > j)
         t(i,j)=max(t[i-1,j], pi+t[i-1,j]-wi)
      Else
         t[i, j]=t[1-i, j]
   }
}
```

Kasu Journal of Computer Science Vol. 1 No. 2 [June, 2024], pp. 412-422
https://doi.org//10.47514/kjcs/2024.1.2.0017
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

```
// Backtracking section

int j = c

For (int i = n; i > 0; i--)

{

   If (t[i,j] != t[i-1,j])

   {

      // Item i was included in the knapsack

      // Add item i to the solution

      // Update j to account for the weight of item i

      j = j - wi

   }

}

Return t[n,c]
```

c. Implementation of the Fully polynomial time Approach:

In the FPTAS algorithm, the goal is to scale down the profits of each item to a more manageable range, which is determined by the accuracy parameter `epsilon`. The scaling is done to ensure that the dynamic programming table can be filled efficiently, and the algorithm remains polynomial in time

```
FPTASalg(p,w,c,i,j,n,t)

//Objective: To obtain the maximum profit of the knapsack

//Objective: To obtain the maximum profit of the knapsack

Input:

  n: number of items

  weights[1..n]: array of item weights

  profits[1..n]: array of item profits

  W: knapsack capacity

  epsilon: accuracy parameter (0 < epsilon < 1)
```

Kasu Journal of Computer Science Vol. 1 No. 2 [June, 2024], pp. 412-422
https://doi.org//10.47514/kjcs/2024.1.2.0017
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

```
Output:
  x[1..n]: binary array indicating selected items
Procedure FPTAS(n, weights, profits, c, epsilon):
  maxProfit = maximum possible total profit (without considering weight constraint)
  scale = epsilon * maxProfit / n
  // Scale the profits
  for i = 1 to n:
    scaledProfit[i] = floor(profits[i] / scale)
  // Dynamic programming table
  t[0..n][0..W]
  // Filling the table
  for i = 1 to n:
    for w = 1 to c:
      if weights[i] <= w:
        t[i][w] = max(t[i-1][w], t[i-1][w-weights[i]] + scaledProfit[i])
      else:
        t[i][w] = t[i-1][w]
  // Backtracking to find selected items
  remainingCapacity = c
  for i = n down to 1:
    if t[i][remainingCapacity] != t[i-1][remainingCapacity]:
      x[i] = 1
      remainingCapacity = remainingCapacity - weights[i]
  // Output the selected items
  return x
```

Kasu Journal of Computer Science Vol. 1 No. 2 [June, 2024], pp. 412-422
https://doi.org//10.47514/kjcs/2024.1.2.0017

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

4. Results and discussion

In this section, we present a comprehensive analysis of three combinatorial algorithms: greedy, dynamic programming, and fully polynomial-time approximation scheme (FPTAS). We provide detailed comparisons in two separate tables (4.1 and 4.2), evaluating their performance based on Halstead complexity metrics and time measures across various capacities for each instance. To evaluate the given code using Halstead Metrics, we calculated the following:

i. Unique Operators(n_1): The number of unique operators and operator symbols in the code.
ii. Unique Operand (n_2): The number of unique variables and constants in the code.
iii. Total Operators (N_1): The total number of operators and operator symbols in the code.
iv. Total Operands (N_2): The total number of variables and constants in the code.

Program vocabulary (n) = n_1+n_2, Program Size/length (N) =N_1+N_2, and Program Volume(V) = $N\log_2(n)$. The results are presented in table 4.1

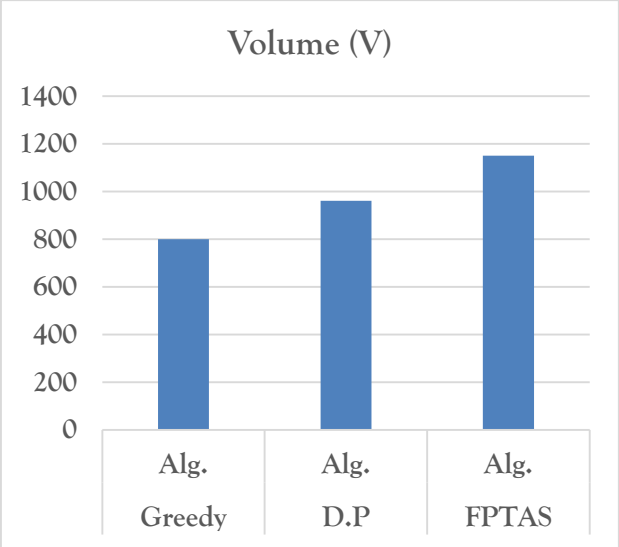
Table 4.1: Programming Complexity measure

Complexity Measure	Greedy Alg.	D.P Alg.	FPTAS Alg.
Input Parameters	n_1=21 N_1=89 n_2=14 N_2=67	n_1=25 N_1=113 n_2=21 N_2=61	n_1=29 N_1=125 n_2=27 N_2=73
Vocabulary (n)	35.00	46.00	56.00
Size (N)	156.00	174.00	198.00
Volume (V)	800.17	961.10	1149.86

From table 4.1, the result of the program complexity computation, based on the Halstead metric it could be observed that the Greedy program version has a volume of 800.17 , dynamic programming version has a volume of 961.10 while FPTAS version has a volume of 1149.86. This implies that FPTAS version is more complex and voluminous than the other two methods. This is depicted graphically in figure 4.1.

Kasu Journal of Computer Science Vol. 1 No. 2 [June, 2024], pp. 412-422
https://doi.org//10.47514/kjcs/2024.1.2.0017

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng



Figures 4.1, comparing the volume of greedy, dynamic programming and FPTAS algorithms

Table 4.2 presents a detailed comparison of execution times, measured in milliseconds, for Greedy, Dynamic Programming (D.P), and Fully Polynomial-Time Approximation Scheme (FPTAS) algorithms, all implemented within the Python programming environment. Notably, as the input size (n) escalates from 5 to 30, significant variations in execution times are observed for both algorithms. Particularly, as the input size increases, the execution times for FPTAS algorithm experience substantial growth. Conversely, Greedy algorithm consistently demonstrate efficient performance, closely followed by Dynamic Programming, exhibiting significantly lower execution times across all input sizes despite the increasing problem complexity. This implies that the Greedy algorithm outperformed the Dynamic Programming and FPTAS strategies in terms of time efficiency in the context of the knapsack problem. Table and Figure 4.2, illustrates the comparison of time complexity among Greedy, Dynamic Programming, and FPTAS algorithms.

Table 4.2: Computational Speed

n	Greedy Alg.	D.P Alg.	FPTAS (where Epsilon=0.5)
5	0.000000	0.088736	0.655174
10	0.000000	0.163459	1.736164
15	0.001000	0.639708	4.811049
20	0.001000	1.037790	7.495216
30	0.001000	1.467000	11.040926

Kasu Journal of Computer Science Vol. 1 No. 2 [June, 2024], pp. 412-422
https://doi.org//10.47514/kjcs/2024.1.2.0017

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

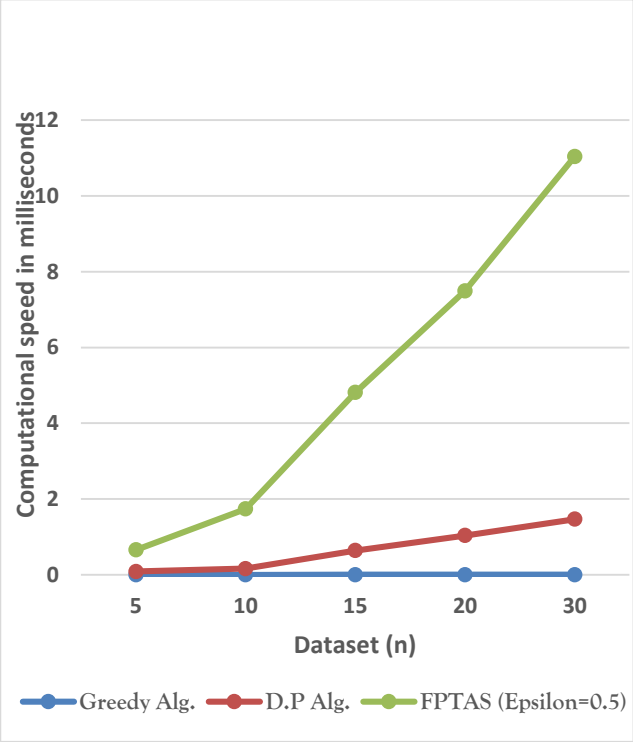


Figure 4.2, Comparing of time complexity among Greedy, Dynamic Programming, and FPTAS algorithms.

Tables 4.1 and 4.2 compare Greedy, Dynamic Programming, and FPTAS algorithms for the knapsack problem. Table 4.1 shows Greedy with complexity volume 800.17, Dynamic Programming at 961.10, and FPTAS highest at 1149.86, indicating FPTAS's greater complexity. Figure 4.1 visualizes these results. Table 4.2 reveals Greedy consistently outperforms Dynamic Programming and FPTAS in execution times (milliseconds) as input size (n) increases from 5 to 30. FPTAS exhibits significant time increases with larger inputs. Figure 4.2 confirms Greed's superior time efficiency, highlighting its practical advantage in solving knapsack problems efficiently.

5. Conclusion

After analyzing the knapsack problem using three distinct heuristics implemented in Python, this study concludes that the Greedy algorithm emerges as the optimal choice for both programmers and theorists in practical scenarios. Greedy algorithms consistently demonstrate superior time efficiency across various problem sizes, making them ideal for applications that prioritize quick decision-making and resource optimization. However, a limitation of the study is its focus primarily on time efficiency and program volume without exploring solution optimality, performance variations across programming languages or hardware, and validation in real-world scenarios.

6. Recommendations for future research:

To advance understanding of the knapsack problem, future research should focus on exploring hybrid approaches, aiming to leverage the unique strengths of each algorithm and alleviate their limitations for

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

improved computational efficiency. Investigating the performance of other programming languages in addressing combinatorial optimization problems would provide valuable insights into the impact of language choice. Additionally, integrating parallel computing techniques could accelerate solution processes for large-scale problem instances.

Reference

Goddard S. (2015). Dynamic programming 0-1 Knapsack problem: http://www.cse.uni.edu/goddard/Courses/CSCE310J.

Kellerer, H., Pferschy U. and Pisinger D. (2004). Knapsack problems. Berlin: Springer. p. 449. ISBN 978-3-540-40286-2. Retrieved 5 May2022.

Namer A., Etawi A. and Fatima T. A. (2020). 0/1 Knapsack Problem: Greedy Vs. Dynamic-Programming. International Journal of Advanced Engineering and Management Research.Vol. 5, No. 02; 2020, ISSN: 2456-3676.

San L. A. (2019). Comparative Study of Dynamic Programming and Greedy Method. International Journal of Computer Applications Technology and Research, Volume 8–Issue 08, 327-330, 2019, ISSN: -2319–8656.DOI:10.7753/IJCATR0808.1007.

Usman B. B., Abdullahi I., and Okeyinka A. E. (2023). Scientific research for smart, secure and sustainable future, in Proceedings of the Third Fonsa Conference, Ibrahim Badamasi Babangida University, Lapai, Niger State.

Wu Y. (2022). Comparison of dynamic programming and greedy algorithms and the way to solve 0-1 knapsack problem. In: Proceedings of the 3rd International Conference on Signal Processing and Machine Learning. DOI: 10.54254/2755- 2721/5/20230666.

Xiaoxi C. (2022). A Comparison of Greedy Algorithm and Dynamic Programming Algorithm. doi.org Saerch 12/09/2022.