Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 7.056

IJCSMC, Vol. 13, Issue. 8, August 2024, pg.60 – 69

Towards Complexity Analysis of Some Combinatorial Algorithms Using Knapsack Model

Bashar Bin Usman[1]; Ibrahim Abdullahi[2]; Okeyinka Aderemi Elisha[3]
Department of Computer Science, Faculty of Natural Sciences, Ibrahim Badamasi Babangida University, Lapai, Niger State, Nigeria
Email: basharbinusman1@gmail.com [1]; ibrojay01@ibbu.edu.ng [2]; aeokeyinka@ibbu.edu.ng [3]

DOI: https://doi.org/10.47760/ijcsmc.2024.v13i08.007

Abstract:
Knapsack problem is a combinatorial optimisation problem which is concerned with finding a solution where an exhaustive search is difficult or even impractical to do. Approximate algorithms, also called heuristics are therefore applied in solving Knapsack problems. The goal of this study was to solve the Knapsack model using three heuristics viz: greedy, dynamic programming, and branch-and-bound strategies. The ultimate objective of our study was to evaluate the time complexities of these heuristics as input sizes become larger. In the bid to get that done, a Knapsack problem was solved using the approximate algorithms, employing the same programming environment, with varying input sizes. The problem was simulated using different input data sizes. A synthesis of the results obtained, that is, the various time complexities, was then carried out and a comparative study of the three algorithms was done. The coefficients of the variables used in the Knapsack model for simulation were generated using random numbers generating algorithm. Our results showed that the greedy algorithm was the most efficient computationally. The second best efficient was the dynamic progranmming algorithm while the least in terms of time complexity is branch-and-bound strategy. It is considered a worthwhile effort to study more metrics of complexity measurement. This is recommended as good enough for further research attention. However, our present study has shown in pragmatic form the computational complexity of the greedy, dynamic programming, and branch-and-bound algorithms.

Keywords: algorithms, heuristics, greedy strategy, complexity, branch-and-bound, dynamic programming

1. Background to the Study

Combinatorial optimization refers to the field focused on finding optimal solutions from a finite set of discrete possibilities. This involves designing algorithms and techniques to solve problems where the goal is to optimize a particular objective, such as minimizing costs or maximizing profits, subject to various constraints. combinatorial optimization has become one of the most trending topics in evolutionary computation today as it has enormous importance in the real world applications (NeumaFriedrichdrich,2017). Combinatorial optimization has been applied in several areas of computing such as auction theory, artificial intelligence, software engineering and so on. This study considers a specific class of optimization problems known as Knapsack Problem.

Knapsack problems are paradigmatic problems in combinatorial optimization which are concerned with finding a solution where an exhaustive search is difficult or even impractical to do. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with valuable items. Common to all versions are set of *n* items, with each item $1 \leq i \leq n$ having an associated profit p_i and weight w_i. The objective is to select some of the items, with maximal total profit, while obeying that the maximum total weight of the chosen items must not exceed c. generally, these coefficients are scaled to become integers, and they are almost always assumed to be positive.

The Knapsack problem is formally stated as follows:

Given n objects and a knapsack of capacity c where object $_i$ has weight $\square_{\square}$ and earns profit $p_{i,}$ find values of $\square_{\square}$ that maximise the total profit.

$$\text{Max} \sum_{i=1}^{n} (x_i p_i)$$

subject to the constraints

$$\sum_{i=1}^{n} (x_i w_i) \leq c,$$

$$0 \leq x_i \leq 1$$

All Knapsack problems are in the domain of NP-hard problems. To solve these problems for optimal or near optimal solution algorithms/heuristics have been developed. They include dynamic programming, branch-and-bound, and greedy algorithms.

The focus of this study was to solve the knapsack model using three heuristics viz: greedy, dynamic programming and branch-and-bound.

This paper unfolds with a Research Motivation in Section 2, followed by an exploration of related literature in Section 3. Section 4, methodology which introduces the conceptual framework, covering random number generation and algorithms. Section 5 presents Results, and Section 6 concludes the paper.

2. Research Motivation

In the realm of computer science and real-world applications, combinatorial problems pose significant challenges, emphasizing the critical role of algorithm selection. Our research is dedicated to a thorough exploration of the knapsack model, wherein we delve into the comparative analysis of three key algorithms: greedy, dynamic programming and branch-and-bound algorithms. Our primary focus is on time complexity, aiming to discern and highlight the efficiency of each method.

By adopting this comprehensive approach, our study seeks to provide valuable insights that empower informed decision-making in the selection of algorithms. This not only contributes to a deeper understanding of the knapsack problem but also holds the potential to advance algorithm design and optimization strategies. Ultimately, our research endeavors to enhance computational solutions, paving the way for more effective approaches to tackling combinatorial problems in both theoretical and practical contexts.

3. Related Works

The knapsack problem, a cornerstone in combinatorial optimization, holds significant implications for real-world applications like resource allocation and project scheduling. This literature review delves into its complexity analysis and explores the efficacy of combinatorial optimization algorithms.

The 0/1 knapsack model, where items possess weights and profits, is a well-studied variant. Fractional knapsack allows item splitting, contrasting with the 0/1 knapsack's prohibition of such splits (Yadav, 2022). The unbounded knapsack, while accepting items only as wholes, permits repeated selection, differing from the 0/1 approach (Maggo, 2022). Additionally, the Multiple Knapsack Problem extends the single knapsack concept, optimizing profits while considering item sizes and bin capacities (Chekuri &Hanna, 2005).

(Vala et al., 2014), analyzed Dynamic Programming and Greedy algorithms for both 0/1 and fractional knapsack problems. Despite similar profits, dynamic programming exhibited superior time efficiency. Limited to an input size of five, the research suggested the promise of dynamic programming in time optimization

(Sampurno et al., 2018) ,addressed the Integer Knapsack problem in freight transportation, employing Greedy and Dynamic Programming Algorithms. Dynamic programming outperformed Greedy, optimizing goods selection through a mobile application. Emphasis was solely on weight

(Aung, 2019), compared Dynamic Programming and Greedy Algorithm for the Knapsack Problem, prioritizing item benefits within capacity. Dynamic programming excelled in time and total value, showcasing parallelism, while the study focused on capacity and time considerations.

(Namer et al., 2020), explored Greedy and Dynamic Programming algorithms, revealing that Dynamic Programming provides superior optimal solutions, while Greedy excels in runtime and space efficiency using Java implementations .

(Chen, 2022), discussed greedy and dynamic programming algorithms for the knapsack problem, highlighting the greedy algorithm's faster but non-optimal results compared to dynamic

programming. The research compared these approaches based on application properties and scope, ultimately favoring the greedy algorithm. The comparison was limited to the time complexity of the two algorithms.

4. Research Methodology

The time complexities of the knapsack model using the three optimization algorithms will be captured. A comparative analysis is carried out in order to find out how efficient the algorithms are, when applied to the knapsack model. Thereafter, the complexities are synthesized, and simulated a few numbers of times. The goal of the synthesis is to derive a more efficient complexity metric for knapsack model.

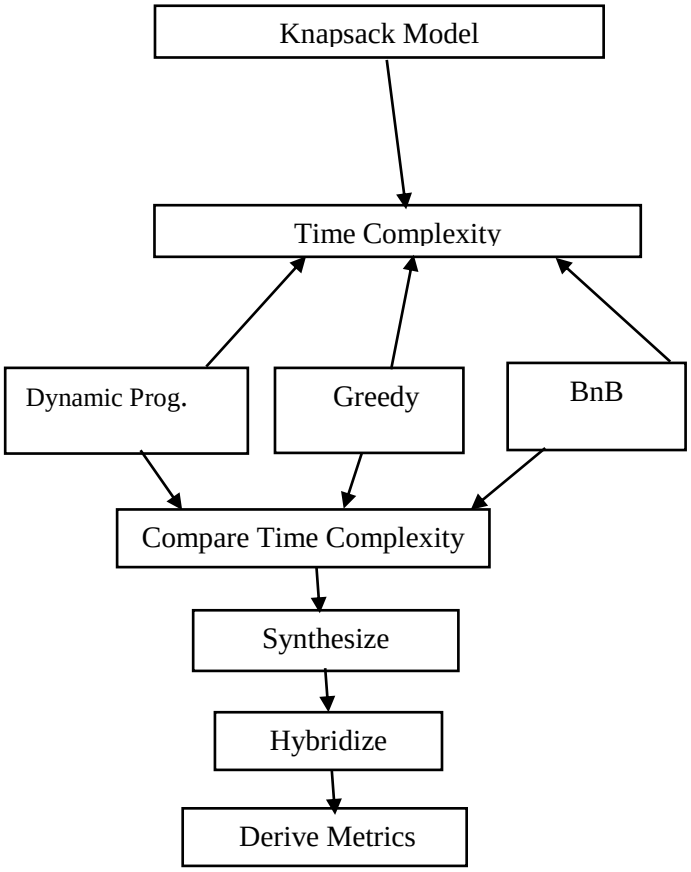

Figure1: Conceptual Framework

4.1 Formulating the Knapsack Problem:

Given a set of n items, each with weight (w_i) and profit ($p_{i)}$, our goal is to determine the ideal quantity of each item for inclusion in a knapsack. The challenge is to ensure that the total weight does not exceed the given capacity limit (c), while maximizing the overall profit. This is expressed through binary variables x_i (i = 1,…,n), where x_i=1 signifies inclusion in the knapsack,

and x_i=0 implies exclusion. The formulation aims to strike a balance between capacity constraints and profit maximization in crafting an efficient solution to the Knapsack Problem.

$$\text{Max} \sum_{i=1}^{n} (x_i p_i)$$

subject to the constraints

$$\sum_{=1}^{n} (x_i w_i) \leq c,$$

$$x_i \in \{0,1\} \quad \forall_i \in \{1,2,3\ldots., n\}$$

4.2 Implementation

The study incorporated four algorithms. The initial algorithm was employed for number generation, while the subsequent three were dedicated to solving the knapsack problem :

Random number algorithm.

Dynamic programming algorithm.

Greedy algorithm.

Branch- and bound algorithm

4.2.1 Random Number Generation Algorithm

The Random Number Generator algorithm produces unpredictable sequences, often utilizing the Residual method (Ri+1 = ARi + B (MOD C)), where A, B, and C are constants, ensuring randomness in numerical sequences.

random_numberalg(A,B,C,R,i)

Input seed, call it R

input constant values of A, B, and C

Input i (number of iteration)

Calculate Ri+1 = ARi + B (MOD C)

repeat step 4

Return random_number

4.2.2 Greedy Algorithm approach

The optimal solution involves making greedy, choices that lead to the optimum. Breaking items into fractional parts, like x_i ($0 \leq xi \leq 1$), ensures precisely filling the knapsack for maximum profit (Yadav, 2022).

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
Endif
Repeat
Return (solution)
```

4.2.3 Dynamic Programming Algorithm Approach

Dynamic Programming (DP) for the knapsack problem involves breaking it into smaller subproblems, solving them iteratively, and storing solutions to avoid redundant calculations. (Bellman,1957).

```
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]
}
}
End
Return [n,c]
```

4.2.4 Branch-and-Bound Algorithm Approach

A branch-and-bound (BnB) algorithm The Branch-and-Bound algorithm, an optimization method for the Knapsack Problem, systematically divides the problem into smaller subproblems, employing bounds to eliminate less promising solutions (Land &Doig,1960)

To solve the knapsack problem in this technique, the upper bound (ub) has to be calculated. This can be computed by adding the total profit of the items that are already selected, say p, the

product of the remaining capacity of the knapsack, c-w, and the best profit-weight ratio, which is pi+1/wi+1. i.e.,

ub= p + (c - w) (pi+1 / wi+1)

Branch -and-bound Alg(p,w,c,i)

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

// Input: c is the capacity of the Knapsack.

n is the number of items.

wi+1 is an array consisting of weight of all n items sorted in decreasing order of profit/weight ratio.

pi+1 is the array consisting of profit of all n items sorted in decreasing order of profit-weight ratio.

i denotes the index pointing to the above arrays (i = 1 initially).

w denotes the current sum of weight (w =0 initially).

p denotes the current sum of profit (p = 0 initially).

//Output: The optimal solution.

while c >= w

do w = w + wi

$p = p + p_i$

i = i + 1

endwhile

ub = p + (C - w) (P_i+1 / W_i+1) // Find the upper bound.

if(ub >= p)

if(i < n)

Brand-and-Boundalg(p, w, c,i)

end if

5. Results and Findings

All the algorithms presented were Implemented in the same environment, the algorithms underwent testing with varied array sizes (n) and consistent capacity. Each algorithm was executed 10 times, and averages were taken. Initial tests on small arrays ensured functionality. Experimental results, tabulated and graphed, display execution times for greedy, dynamic programming, and branch-and-bound algorithms across different input sizes (n).

Table 1: Comparison for Greedy, Dynamic programming, and Branch-and-bound algorithms

n	Greedy Alg.	D.P Alg.	BnB Alg.
5	0.000000	0.088736	0.007170000
10	0.000000	0.163459	0.205530000
15	0.001000	0.639708	6.618620000
20	0.001000	1.037790	119.1072200
25	0.001000	1.408670	3698.558440

30	0.001000	1.467000	124747.4591
35	0.002000	2.338020	7351382.174

Analyzing three algorithms in the context of the knapsack problem reveals a trade-off in computational efficiency. The greedy algorithm excels in speed, followed by dynamic programming, while Branch-and-Bound grows exponentially, surpassing both greedy and dynamic programming approaches in time complexity.

Graphical Representations of Algorithm Performance

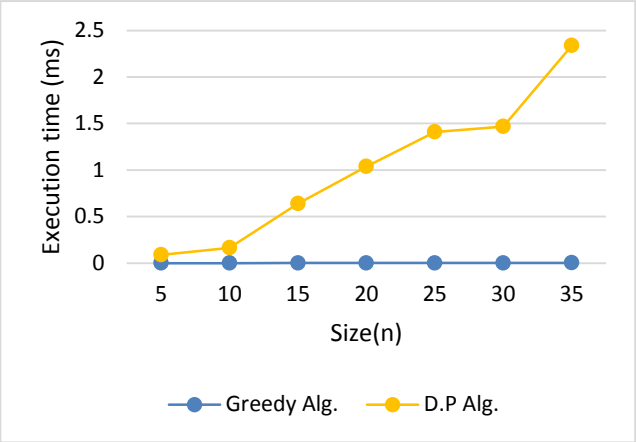

Figure 2: Comparison between greedy and dynamic programming algorithms

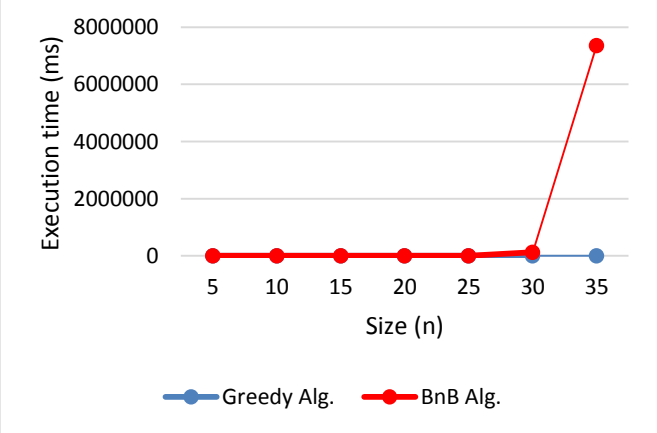

Figure 3: Comparison between greedy, and branch-and-bound algorithms

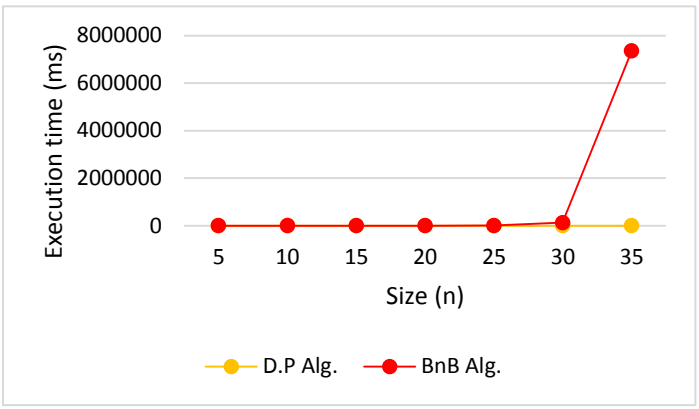

Figure 4: Comparison between dynamic programming, and branch-and-bound algorithms

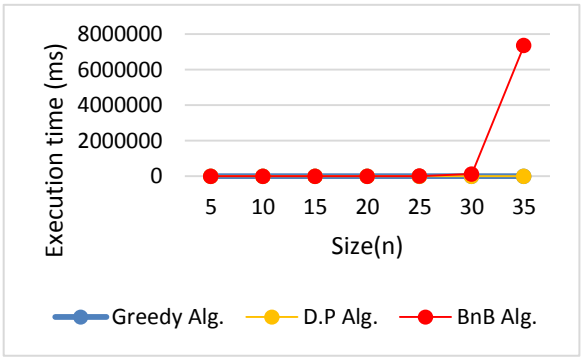

Figure 5: Comparison of greedy, dynamic programming, and branch-and-bound algorithms

The chart conceals Greedy and dynamic programming algorithms due to branch-and-bound's growth in execution times, causing a vast scale difference and hindering visual differentiation.

The figure 5 demonstrate the effectiveness of the algorithms in terms of solving the knapsack model. We can observe that greedy algorithm required minimum time to implement the knapsack model, followed by dynamic programming algorithm while the least in terms of time complexity is branch-and-bound strategy since it's complexity grows more exponentially than those of the remaining two heuristics.

Based on this, we can deduce that greedy algorithm is more efficient than the two algorithms concerning time when dealing tackling the knapsack problem.

6. Conclusion

Our study contributes to a better understanding of the complexity landscape of the knapsack problem and the efficiency of various combinatorial optimization algorithms. The results offer valuable insights into which algorithm to choose based on problem characteristics, and they provide guidance for practitioners dealing with real-world knapsack instances. Our future research would explore hybrid approaches by combining the different algorithms in order to exploit their strengths and mitigate their weaknesses.

References

[1]. Bellman, R. E., (1957). Dynamic Programming. Princeton University Press.

[2]. Chekuri C., (2005), A PTAS for the multiple knapsack problem, Society for Industrial and applied mathematics: 213-222

[3]. Land, A. H., & Doig, A. G. (1960). An automatic method of solving discrete programming problems. Econometrica, 28(3), 497-520.

[4]. Namer A. A ., and Fatimah 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

[5]. Maggo R., (2022) Unbounded knapsack. https://www.codingninjas.com/codestudio/library/unbounded-knapsacks search on 22/2022

[6]. Friedrich T. and Neumann F., (2017). What's hot in evolutionary computation," Conference on Artificial Intelligence (AAAI), pages 5064–5066.

[7]. Chen X., (2022).A Comparison of Greedy Algorithm and Dynamic Programming Algorithm. doi.org Saerch 12/09/2022

[8]. Yadav P. (2022) Fractional Knapsack Problem. https://www.scaler.com/topics/fractional-knapsack-problem seachon 21/12/2o22

[9]. Sampurno G. I. , Endang S., and Alamsyah(2018).Comparison of Dynamic Programming Algorithm and Greedy Algorithm on Integer Knapsack Problem in Freight Transportation. Scientific Journal of Informatics Vol. 5, No. 1; p-ISSN 2407-7658

[10]. San L. A, (2019).Comparative Study of Dynamic Programming and Greedy Method. International Journal of Computer Applications Technology and Research Vol. 8, ISSN:-2319–8656

[11]. Vala J., Dhara M., & Aymit P.(2014). Comparative Analysis of Dynamic and Greedy Approaches for Dynamic Programming. International Journal of Modern Trends in Engineering and Research e-ISSN: 2349-9745

