American Journal of Applied Mathematics and Statistics, 2024, Vol. 12, No. 3, 66-69
Available online at http://pubs.sciepub.com/ajams/12/3/4
Published by Science and Education Publishing
DOI:10.12691/ajams-12-3-4

Analysis of Complexity of Some Combinatorial Algorithms Using Halstead Metrics and Time Measures

Bashar Bin Usman*, Ibrahim Abdullahi, Okeyinka Aderemi Elisha

Department of Computer Science, Faculty of Natural Sciences, Ibrahim Badamasi Babangida University, Lapai, Niger State, Nigeria
*Corresponding author: basharbinusman1@gmail.com

Received July 15, 2024; Revised August 18, 2024; Accepted August 25, 2024

Abstract The Knapsack problem is a combinatorial optimization problem for which finding an exact solution using exhaustive search methods is impractical. Therefore, approximate algorithms are usually considered when tackling this optimization problem. The goal of this study was to analyze the complexity of the Knapsack problem using three approximate algorithms: greedy, dynamic programming, and branch-and-bound methods. To achieve this, we measured the Halstead metrics and computational time complexity of these three algorithms. Our methodology involved solving a hypothetical Knapsack problem using the three algorithms in the same programming environment. Experiments were conducted with varying input datasets, and the time complexities of the algorithms were recorded for each experiment. The average time complexities were computed at the end of the experiments. Additionally, we calculated the Halstead metrics required for the study. Both the Halstead metrics and computational time metrics for the three algorithms were then compared. Our findings showed that the greedy approach was the most efficient heuristic, followed closely by the dynamic programming method, with the branch-and-bound algorithm being the least efficient. Future work will incorporate real-world data with specific item characteristics to enhance the relevance and applicability of the findings. Additionally, it will explore hybrid approaches that combine these algorithms to leverage their respective strengths and effectively address their weaknesses.

Keywords: *heuristics, complexity, knapsack problem, greedy, branch-and-bound, dynamic programming, halstead metrics*

Cite This Article: Bashar Bin Usman, Ibrahim Abdullahi, and Okeyinka Aderemi Elisha, "Analysis of Complexity of Some Combinatorial Algorithms Using Halstead Metrics and Time Measures." *American Journal of Cancer Prevention*, vol. 12, no. 3 (2024): 66-69. doi: 10.12691/ajams-12-3-4.

1. Introduction

The Knapsack problem (kp), is a renowned combinatorial optimization challenge, is prevalent in various fields such as business, finance, logistics, and transportation. Due to its NP-Hard nature, finding exact solutions through exhaustive methods is often impractical [1]. Suppose we are given a set of n items, each with a weight wi and associated profit pi , determine the number of each item to include in a knapsack so that the total (weight) capacity c is less than or equal to a given limit and the total profit pi is as large as possible. KP can be formulated as follows:

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

subject to the constraints

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

$$0 \leq x_i \leq 1$$

Let pi be the profit of item i, wi be the weight of item i, c be the capacity of the knapsack, and n be the total number of items in the knapsack problem (KP). The variable xi, which can be either 0 or 1, indicates whether item i is selected (1) or not (0). For simplicity, it is assumed that all items are arranged in non-increasing order of their efficiency, meaning: (pi/wi) ≥ pi+1/wi+1 (i=1,2,..., n-1)This ensures that items are sorted by their profit-to-weight ratio in descending order.

The knapsack problem extends to classes of problems in business, network optimization, and production planning [2]. This study focuses on solving the knapsack problem using three heuristics: greedy, dynamic programming, and branch-and-bound algorithms, with emphasis on Halstead metrics and computational time measures.

This paper is organized in the following manner: Section 2 delves into the Research Motivation, emphasizing the need for comprehensive algorithmic evaluations. Section 3 provides a thorough overview of related literature, highlighting key findings from existing studies. In Section 4, the conceptual framework is presented, discussing random number generation and the three algorithms. Section 5 details the Experimental Results, and Section 6 outlines the Conclusion,

summarizing the study's contributions, suggesting avenues for future research and 7 conflict of interest statement.

2. Research Motivation

The complexity analysis of combinatorial algorithms, employing Halstead metrics and time measures, addresses the crucial need for deeper insights into their performance characteristics. Combinatorial problems are widespread, and choosing the most suitable algorithm is paramount [3]. This study's dual-pronged approach, evaluating both code complexity and execution efficiency, contributes to informed decision-making in algorithm selection. By analyzing the complexity of the knapsack model using greedy, dynamic programming, and branch-and-bound methods, this research aims to advance algorithm design and optimization, enhancing computational solutions for diverse complex problems.

3. Related Works

Extensive research on the knapsack problem reveals its significance in real-world applications. Studies such as [4,5,6,7], and [8] compare dynamic programming and greedy algorithms in various contexts, considering factors such as profit, efficiency, weight optimization, and runtime. These comparative analyses contribute valuable insights into algorithmic performance.Their study's limitations include focusing solely on container capacity, using only five input sizes, comparing weight alone, implementing in Java JDK 8.0 with random item weights, and restricting comparison to two algorithms based on time complexity.

While the aforementioned studies offer valuable insights into the algorithmic efficiencies for the knapsack problem, there is a potential research gap in exploring more advanced algorithms beyond Dynamic Programming and Greedy approaches. Specifically, there is an opportunity to investigate the effectiveness of advanced algorithm like branch-and-bound.

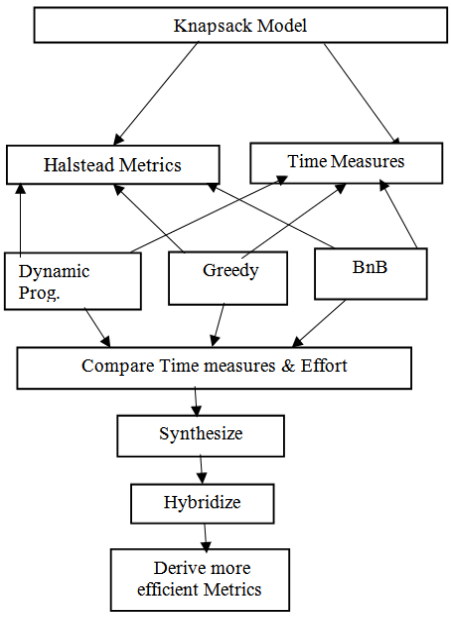

Figure 1. Methodological Framework

4. Research Methodology

The research involves a comprehensive examination of the complexities of the knapsack model using three optimization algorithms: greedy, dynamic programming, and branch-and-bound. The primary objective is to conduct a comparative analysis to assess the efficiency of these algorithms. The study synthesizes complexities and subjects them to experimental runs, utilizing Halstead measures and time measures for an assessment of algorithmic performance. The overarching aim is to establish a more efficient complexity metric for the knapsack model.

4.1. Random Number Generator

The Random Number Generator is an algorithm that produces a sequence of numbers characterized by unpredictability and the absence of any recognizable pattern. While various arithmetic operations can be employed, the Residual method is a commonly used approach. This method is defined by the recurrence relation Ri+1 = (A * Ri + B) % C, where A, B, and C are constants, and Ri and Ri+1 represent the ith and (i+1)th random numbers.

The algorithm for generating random numbers is outlined below:

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

1. Input the seed, denoted as R.
2. Input constant values for A, B, and C.
3. Input the number of iterations, denoted as i.
4. Initialize a variable, let's call it current_random, with the value R.
5. For each iteration from 1 to i:
   a. Calculate current_random=(A*current_random + B) % C.
   b. Update current_random with the new value.
6. Return the final value of current_random.

Here are the three algorithms for solving the knapsack problem

4.2. Greedy Algorithm

The Greedy approach sequentially assesses inputs, considering their suitability based on a predefined order. If incorporating the next input compromises the ongoing optimal solution, it's excluded. The selection process, guided by an optimization criterion, yields multiple viable solutions, with the optimal one determined later [9]

Greedy (p,w,c,i)

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

// input: list of items, each with a profit p_i and a weight w_i

// 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)

4.3. Dynamic programming Algorithm

Is an optimization method that efficiently solves the knapsack problem by breaking it into smaller subproblems. It constructs a table to store the maximum value that can be achieved at each capacity, considering previous subproblem solutions. By iteratively filling the table, it ensures optimal solutions for each subproblem, leading to the determination 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 w_i
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
$(w_i > j)$
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]

4.4. Branch-and-bound Algorithm

A Branch-and-Bound (BnB) algorithm:
This systematically explores candidate solutions by constructing a decision tree. Nodes represent partial solutions within upper and lower bounds. Efficiently choosing nodes at each level leads to rapid identification of the optimal solution. Finding upper bounds involves local optimization or selecting points in the search space. For the knapsack problem, the upper bound (ub) considers selected items' profit, remaining capacity, and the best profit-weight ratio: ub = p + (c - w) (p_{i+1} / w_{i+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.
w_{i+1} is an array consisting of weight of all n items sorted in decreasing order of profit/weight ratio.
p_{i+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 + w_i
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 Findiings

In this section, we provide details of the Comparative analysis of the three combinatorial algorithms namely, greedy, dynamic programming and branch-and-bound algorithms using both time measures and Halstead complexity metrics to analyze the performance of these algorithms.while keeping the capacity size constant for each instances. additionally, the analysis used synthetic data generated by a random generator to maintain consistency and control over the variables.

Table 1. Time measures 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
Average	0.000857	1.020483	1068564.876

The Table 1 shows the execution times in milliseconds for the three different algorithms applied to the knapsack problem with varying problem sizes (number of items, n).These algorithms were implemented in the same programming environment.

Table 2. Halstead metrics for Greedy, Dynamic programming, and Branch-and-bound algorithms

Comlexity Measure	Greedy Alg.	D.P Alg.	BnB Alg.
Vocabulary(n)	61.00000	54.00000	82.00000
Size/Length(N)	320.0000	320.0000	422.0000
Volume(V)	1897.600	1841.600	2683.900
Difficulty(D)	106.0400	114.2000	87.78000
Effort(E)	201221.5	210310.7	235592.7
line-of-code	89	78	110

In comparing three algorithms for solving the knapsack problem using Halstead metrics, Branch and Bound exhibited the highest overall complexity and effort, with the most lines of code. The Greedy Algorithm had the highest vocabulary, while Dynamic Programming fell between the other two in most metrics.

Table 3. Time measures and Halstead metric (effort)

S/N	Algs	Time measures	Halstead metric (effort)
1	Greedy	O.000857	201221.5
2	Dynamic prog.	1.020483	210310.7
3	BnB	1068564.876	235592.7

Table 3 comprises the average experimental time complexity for each algorithm and the overall effort assessed through Halstead metrics.

Time measures assess a program's runtime performance, while Halstead metrics analyze code complexity and development effort

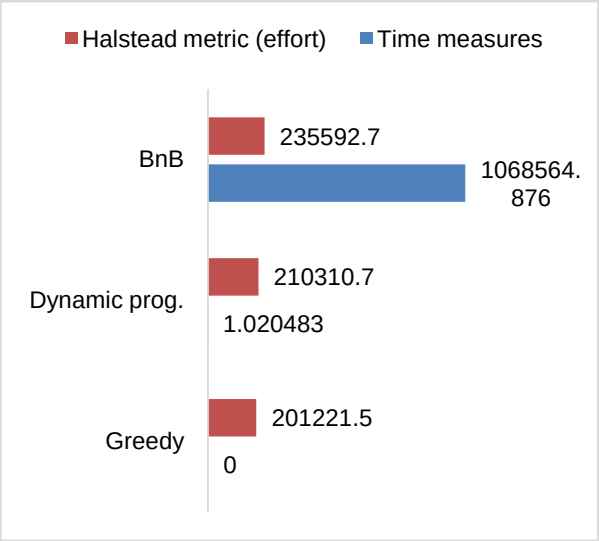

Figure 2. Comparing time measures and Halstead metric (effort)

The results presented in Figure 1 showcase a comparison between time measures and Halstead metrics evaluations to assess the effectiveness of different algorithms in tackling the knapsack problem. It's evident that the greedy algorithm demands the least amount of time and effort for implementing the knapsack model followed by dynamic programming algorithm. Conversely, the branch-and-bound strategy exhibits the longest execution time and requires the most effort.

Based on this we can deduce that the greedy approach was the most efficient heuristic, followed closely by the dynamic programming method, and the least efficient is the branch-and-bound algorithm in terms of time efficiency.

6. Conclusion

This study enhances our comprehension of the complexity landscape associated with the knapsack problem and the efficiency of various combinatorial optimization algorithms, the combination of time measures and Halstead complexity metrics for solving knapsack problem provides a more holistic understanding of algorithmic efficiency, allowing researchers and practitioners to identify potential areas for code optimization and make effective technical and managerial decisions Future work will incorporate real-world data with specific item characteristics to enhance the relevance and applicability of the findings. Additionally, it will explore hybrid approaches that combine these algorithms to leverage their respective strengths and effectively address their weaknesses.

7. Conflict of Interest Statement

We declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] Elizabeth L. How the Mathematical Conundrum Called the 'Knapsack Problem Is All Around Us. https://www.smithsonianmag.com/science-nature/why-knapsack-problem-all-around-us-180974333/2020; search on 25/09/23.

[2] Ingariola, G, & Korsh, J. A general algorithm for one dimensional knapsack problems, Operation Research 1973;25(5). Friedrich T. and Neumann F. What's hot in evolutionary computation, Conference on Artificial Intelligence (AAAI), 2017;5064–5066.

[3] Vala J, Monaka D, Pandya J. Comparative Analysis of Dynamic and Greedy Approaches for Dynamic Programming. Int J Mod Trends Eng Res. 2014; 5.

[4] Omotosho O. I. & Okeyinka A.E., (2015). Comparative analysis of the greedy method and dynamic programming in solving the knapsack problem. Global Journal of Advanced Engineering Technologies and Sciences .

[5] Ameen S, & Azzam S. Comparing between different approaches to solve the 0/1 Knapsack problem. IJCSNS International Journal of Computer Science and Network Security, 16(7), 2016.

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

[7] Namer AA, Fatimah TA. 0/1 knapsack problem: greedy vs. dynamic-programming. International Journal of Advanced Engineering and Management Research 2020; 5(02).

[8] Chen X. A Comparison of Greedy Algorithm and Dynamic Programming Algorithm. doi.org 2022; Saerch 12/09/2022.

[9] Oluyinka IO, Okeyinka AE. Comparative analysis of the greedy method and dynamic programming in solving the knapsack problem. Global Journal of Advanced Engineering Technologies and Sciences. 2015.