ISSN No:-2456-2165

An Explanatory Comparison of Brute Force and Dynamic Programming Techniques to Solve the 0-1 Knapsack Problem

O.A Balogun [1]
M.A Dosunmu[2]
I.O Ibidapo[2]
[1,2]Department of Computer Science and Information Technology, Bells University of Technology, Ota, Ogun State, Nigeria.

Abstract:- During examination periods in schools, the demand on the use of examination rooms usually increases thus necessitating an optimal allocation of such rooms. This work seeks to explore an optimal way of allocating examination rooms by modeling the problem as a 0-1 Knapsack problem. Two approaches namely the Brute force and Dynamic programming techniques are used and implemented in Excel® and NetBeans® environments. The obtained results showed that dynamic programming approach yields a more optimal result than the Brute force implementation.

Keywords:- *Knapsack, Dynamic Algorithm, Brute Force.*

I. INTRODUCTION

In nature, getting a result from a process involves usually a series of steps. In some cases, some steps are repeated while others occur once. These steps, also known as algorithms have been studied extensively by computer scientists and have helped to frame modern day computing and its related technologies. [14] suggests that we think of an algorithm as a recipe that describes the exact steps needed for a computer to solve a problem or reach a goal. In other words, it can be defined as a detailed step-by-step method for solving a problem by using a computer or a sequence of unambiguous instructions for solving a problem in a finite amount of time.

A major part of human existence is the idea of getting the most value based on a limit at the smallest cost. An example is a person going shopping for clothes. Naturally, the person wants to get high quality clothes at the best possible cost. This situation is called a Knapsack problem in computer science and have been studied extensively and applied to solve problems in the Financial, Manufacturing, and other industries at large. The efficiency of an algorithm is measured with respect to the time required execute it and the amount of computational space used. Algorithms can also be sorted into sequential, parallel, and exact and approximate algorithms.

Usually, in solving robust computing problems (problems that are dynamic in nature), there is usually more than a correct solution. This helps us to pick the most favourable solution (an optimal solution), with respect to time and space, that makes use of lesser resources and gives maximum output gain with respect to time and space. Dynamic programming is a well-suited technique for optimization problems because it solves problems by combining solutions of solved sub-problems.

According to [8], Brute force algorithm, on the other hand, is a type of algorithm that tries a large number of patterns to solve a problem and rely on raw computing power to achieve this in some cases.

Within every educational system, a period to test the knowledge of students based on what they have been taught must occur. In most formal educational setting, this period usually comes after rigorous class sessions. Not all citadels of learning have the structural resources to accommodate a combination of the number of students that can sit for different courses within the same time period. This work aims to find an optimal solution to this problem.

In this work, the data was modelled like that of typical University with a limited number of halls which invariably elongates the examination period. Given the number of courses that a typical university runs, this works uses Brute force and Dynamic programming techniques to find an optimal solution to a 0-1 knapsack problem in the configuration of number of students that can make use of a particular hall at the same time.

In the 0-1 knapsack problem, each item (in this case, all students sitting for a particular course) must be put entirely in the knapsack or not included at all, hence, the 0-1 connotation. The number of resources is limited to one hall because of the weight constraint of the knapsack problem. For the purpose of this work, we name this hall Main Hall.

II. LITERATURE REVIEW

Algorithms help us put computational and resources requirement for solving a problem in the right perspective. Various works like [2] and [5] discuss the strength and weaknesses of algorithms which ultimately influence choice. Generally, factors that influence an algorithm's choice are the number of items to be worked on, the type of algorithimic activity that would be carried out on such items, restrictions on the items, and the kind of storage device that would be used.

[1] posits that rather than solving overlapping subproblems repeatedly, dynamic programming suggests solving each of the smaller sub problems only once and

recording the results in a table from which we can then obtain a solution to the original problem. The precding statement suggests that dynamic programming help to make use of little amount of computational resources because instead of solving all sub-problems, it makes use of the result of previously solved problem to solve other problem withing the problem cycle phases. This also helps to save time which is the other primary focus of efficient algorithms.

Consider a person holding a sack that can contain a set number of items in a store. In a typical store, there are various items with different values and weights. The knapsack algorithm's goal is to fit into the sack, a combination of items, that are most valuable that will not exceed the capacity of the sack. From the illustration, we see that the knapsack problem is a combinatorial optimization problem by nature because it chooses the possible items that do not exceed the preset values. According [6], the 0-1 knapsack problem is the most common problem being solved. This Knapsack problem restricts the number of the copies of items to 0 or 1 and is represented by the formula:
maximize $\sum_{i=1}^{n} v_i\, x_i$ subject to $\sum_{i=1}^{n} w_i\, x_i \leq W and\; x_i \in \{0,1\}$ (1)

Given a set of items numbered from 1 up to n , each with a weight w_i and a value v_i along with a maximum weight capacity W. The bounded Knapsack problem removes the restriction of having just an instance of an item, i.e.,

maximize $\sum_{i=1}^{n} v_i\, x_i$ subject to $\sum_{i=1}^{n} w_i\, x_i \leq W and\; 0 \leq x_i \leq c$ (2)

Although it removes the situation of one copy of an item in the collection, it restricts the number, x_i, to a maximum non negative integer, c.

The unbounded knapsack poblem, while removing restrictions on the number of items, also allows no upper bound on the items, i.e
maximize $\sum_{i=1}^{n} v_i\, x_i$ subject to $\sum_{i=1}^{n} w_i\, x_i \leq W and\; x_i \geq 0$ (3)

[13] show that the general class of questions for which some algorithm can provide an answer in polynomial time is called "class P" or just "P". This means that finding the answer to some questions can not be gotten quickly but the solutions presented can be can be verified based on the information given about the solution. The class of questions for which an answer can be verified in polynomial time is called NP, which stands for "nondeterministic polynomial time". Knapsack problems fall into class NP problems.

[12] presented the greedy algorithm, dynamic programming, and bound technique methods of solving the knapsack problem. They altered the Greedy technique to work for a 0-1 Knapsack problem. They made use of a recursive method for the Branch and Bound technique to expedite the computations and to reduce the memory consumed.

Their work showed that the Greedy algorithm was the most efficient but it is inappropriate under certain conditions because it does not always result in the most optimal solution.The dynamic programming technique, on the othe hand, has proved to be very efficient in terms of number of computations for lesser capacities, but as the capacity of the knapsack increases, this technique proves to be inefficient. The memory utilized by this technique is also the highest among the three approaches considered. Their result showed that the most efficient approach for the Knapsack problem is the Recursive Branch and Bound technique because it is simple and is easy to apply, and can be applied to solve the knapsack

[3] made use of metaheuristics based on neural-networks paradigm for solving the Multidimensional Knapsack Problem (MKP). They show how Neural Networks can be incorporated to solve domain specific problem and provided a mathematical formulation for their algorithm. Making use of the Augmented Neural Networks (AugNN) which takes advantage of both the greedy-heuristic approach and the iterative local-search approach they were able to exploit AugNN by utilizing proven base heuristics to exploit the problem and domain-specific structure and then iteratively search the local neighbourhood in a somewhat random yet guided manner in an effort to improve upon the initial solution.

Their result showed that AugNN meta-heuristic performed favourably in terms of both solution time and quality on a well-known set of difficult benchmark instances and that relative to the other techniques, their technique showed simplicity and provided very favourable results.

[7] proposed a new hybrid heuristic approach that combines the Quantum Particle Swarm Optimization technique with a local search method to solve the Multidimensional Knapsack problem while incorporating a heuristic repair operator that uses problem-specific knowledge instead of the penalty function technique commonly used for constrained problems. Their results showed that on a wide set of benchmark problems, their method demonstrated competitiveness compared to other well established state-of-the-art heuristic methods.

[10] presented an Improved Fruit Fly Optimization Algorithm (IFFOA) for solving the Multidimensional Knapsack Problem (MKP). Parallel search algorithm was employed to balance exploitation and exploration and to make full use of swarm intelligence. A modified Harmony Search Algorithm (MHS) was proposed and applied to add cooperation among swarms in IFFOA. Their work involved extensive numerical simulations and comparisons with other state-of-the-art algorithms. Their result show that their algorithm is a an effective alternative for solving the MKP.

[11] made use of the Knapsack problem to solve advertorial issues in a radio station. With a pile of adverts that needed to be aired, given a limited amount of time, they developed a software that optimally solved the selection problem. Their result showed that out of 900 seconds of

alloted time for adverts, their algorithm was able to optimally make use of 890 seconds.

[4] presented a novel Binary Monarch Butterfly Optimization (BMBO) method, intended for addressing the 0–1 knapsack problem (0–1 KP). Two tuples, consisting of real-valued vectors and binary vectors were used to represent the monarch butterfly individuals in BMBO. Real-valued vectors constituted the search space, whereas binary vectors formed the solution space. Three kinds of individual allocation schemes were tested in order to achieve better performance. Toward revising the infeasible solutions and optimizing the feasible ones, a novel repair operator, based on greedy strategy, was employed.

BMBO was verified and compared with BABC (Binary Version of Artificial Bee Colony Algorithm), BCS (Binary Cuckoo Search algorithm), BDE (Binary differential evolution based memetic algorithm) and GA (Genetic algorithm) on three types of 0–1 KP instances. Their experimental results showed that BMBO outperformed the other four methods in terms of search accuracy, convergent speed and numerical stability.

The comparative study of the BMBO with four state-of-the-art classical algorithms pointed toward the superiority of the former in terms of search accuracy, convergent capability and stability in solving the 0–1 KP, especially for the high-dimensional instances.

[9] made use of Cohort Intelligence (CI) Algorithm to solving 0–1 Knapsack problem. They showed that learning with CI refers to a cohort candidate's effort to self-supervise its own behaviour and further adapt to the behavior of the other candidate which it intends to follow which in turn makes every candidate to improve/evolve its behaviour and eventually the entire cohort behavior. This approach was tested by using it to solve an NP-hard combinatorial problem such as Knapsack Problem (KP). Several cases of the 0–1 KP were solved. Their results showed that the CI methodology produced satisfactory results with reasonable computational cost. Furthermore, according to the solution comparison of CI with other contemporary methods, they posited that the CI solution was comparable and for some problems even better than the other methods. The CI methodology was therefore validated and the self-supervising nature of the cohort candidates was successfully demonstrated along with their ability to learn and improve qualities which further improved their individual behaviour.

III. METHODOLOGY

A. *The Knapsack Problem*

Given a number of items ***a***, i.e.a_1, a_2,…,a_n, with each item having a value,***v***,v_1, v_2,…,v_n and a weight,***w***,w_1, w,…,w_n the 0-1 knapsack problem seeks to find the number of items that can wholly fit into a sack subject to the overall weight, ***W***, that the sack can carry given that ***W*** ≥ 0 such that ***a***'⊆ ***a***

The above statement can be mathermatically represented by
Maximise

$$\sum_{i \in a'} v_i \ldots \ldots \ldots 3.1$$

Subject to

$$\sum_{i \in a'} w_i \leq W \ldots \ldots \ldots 3.2$$

As earlier stated, we will be calling the examination room, Main hall. Main hall has a capacity (i.e. *W*) of 528. The list contains 10 courses i.e. c_1, c_2,…,c_{10} with values (course units) betweeen 2 and 3, as listed in Table 1:

Table 1: Courses weight and value distibution

Courses	weight (*w*)	Value (*v*)
c_1	476	3
c_2	65	2
c_3	481	3
c_4	31	3
c_5	521	3
c_6	62	3
c_7	135	3
c_8	321	2
c_9	124	3
c_{10}	197	2

Capacity =528

As earlier stated , the Knapsack problem is of combinatorial optimization laying emphasis that the solution to the problem will not necessarily follow a particular order i.e. combination does not emphasise the order in which the problem is to be solved.

$$c_{(n,r)} = \frac{n!}{(n-r)!r!} \ldots \ldots \ldots 3.3$$

B. *Brute force algorithm application*

- Compute all the subsets of the list to give 2^n number of subsets.
- Find the sum of the weights in each set and note those that do not increase by the weight capacity. In this situation, sum weights in each subset exceed the capacity.
- Repeat step 1 for each subset until a sub list set of weight does not increase by *W*.
- Sum the values of the sub lists that satisfy the condition in (iii).
- Select the highest value as the answer to the problem.

C. *Dynamic programming application*

General approach

- Characterize the structure of an optimal solution
- Recursively define the value of an optimal solution
- Compute the value of an optimal solution in a bottom-up fashion
- Construct an optimal solution from computed information

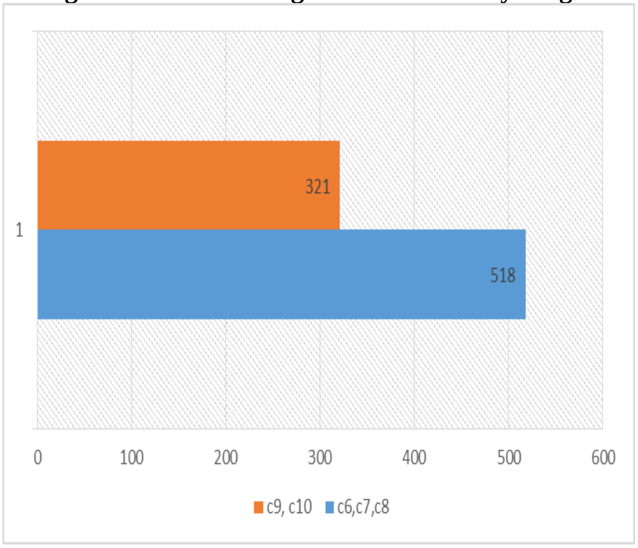
IV. RESULT AND DISCUSSION

In the process of finding optimal solutions, we made use of Microsoft Excel and NetBeans IDE version 8.2. The results are presented in the figures below

- *Brute force algorithm*

Using a top-down approach, the brute force algorithms gave the following result

Fig 4.1: Brute force algorithm solution by weight



Making use of the top two results, the Brute force algorithm gave a combined weight value of 321 (from courses, c_9,c_{10}) and 518 (from courses c_6, c_7,c_8).

Fig 4.2: Brute force algorithm solution by value

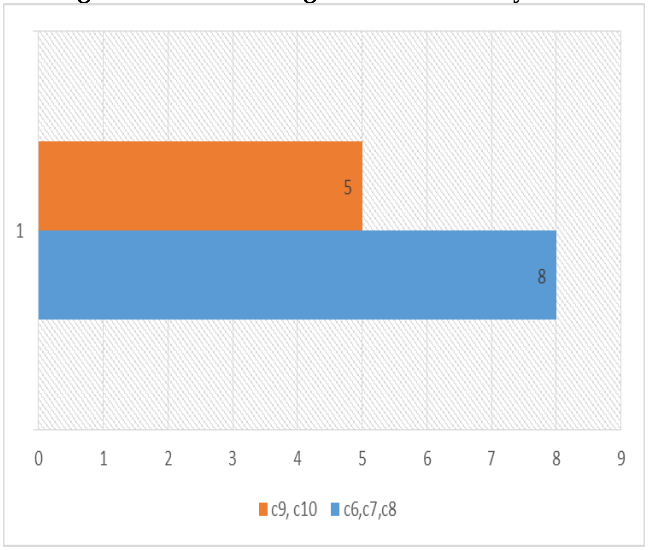
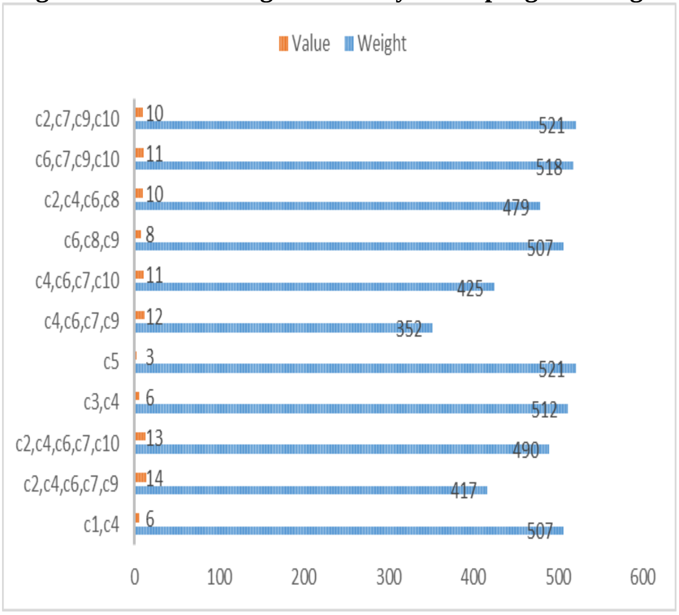


Chart 2 shows that coursesc_6, c_7,c_8) have the most value, thereby making it the optimal solution for brute force algorithm.

Fig 4.3: Chart showing result of dynamic programming



Our results (From chart 3), show that the combination c_2, c_7,c_9, c_{10} gives the highest weight i.e. 7 spare spaces will remain in Main hall but has has a unit value of 10. c_2, c_6,c_9, c_{10} has a weight of 518 i.e. 10 les than the capacity of main hall but has a higher value of 11that that of c_2, c_7,c_9, c_{10}

Our result shows that the optimal solution to this conundrum are the combinations c_2, c_4,c_6, c_7 and c_9 because it has the highest value of 14 and a weight of 417. A potential candidate for the optimal solution are the combinations c_2, c_4,c_6, c_7 and c_{10} because it has a weight of 490 although it carries a value of 13.

Both solutions

- Remove half of the courses to be written in Main hall thereby freeing up the hall largely for other courses.
- Are not too far away from the capacity number of the hall.
- Analysis also show that Brute force algorithm far less number of steps and less use of system computational resources to arrive at its optimal solution than that of dynamic programing

V. CONCLUSION AND RECOMMENDATION

This work took a look into Brute force and Dynamic programming to solve a sitting arrangement conundrum modelledlike the 0-1 Knapsack problem. After data generation and algorithms implementation, we were able to get optimal solutions to the problem which gave an insight into the requirements of both algorithms.

We see that Brute force got the job done thereby laying credence to its efficiency but it also shows that it was incapable of solving the problem optimally to fit into a real-world scenario i.e., computationally, the Brute force approach is not economically realistic.

Dynamic programming on the other hand was able to provide as many optimal options that may be suitable for different situations within the context of solving the problem. We therefore recommend that more data should be used in order to appropriately judge the efficiencies of both algorithms in scenarios like this.

REFERENCES

[1]. BrainKart.com. (2018). *Dynamic Programming.* Retrieved from BrainKart.com: https://www.brainkart.com/article/Dynamic-Programming_8041/

[2]. Chatterjee, J. m. (2015). *A study of the factors considered when choosing an appropiate data mining algorithm.* Retrieved September 24, 2019, from LinkedIn Slideshare: https://www.slideshare.net/jyotirmoyc1/a-study-on-the-factors-considered-when-choosing-an-appropriate-data-mining-algorithm

[3]. Deane, J., & Agarwal, A. (n.d). Neural metaheuristics for the multidimensional knapsack. Retrieved 2019, from https://pdfs.semanticscholar.org/2f57/1ae946c865b4e98e0a980c31806a686b4621.pdf

[4]. Feng, Y., Wang, G.-G., Deb, S., Lu, M., & Zhao, X.-J. (2015). Solving 0–1 knapsack problem by a novel binary monarch. *Neural Comput & Applic*. doi:DOI 10.1007/s00521-015-2135-1

[5]. Folorunsho, O., vincent, O. R., & Salako, O. (2010). An exploratory study of critical factors affectning the effeciency of sorting techniques (shell, heap and treap). *Annals computer science series*. Retrieved September 24, 2019, from https://pdfs.semanticscholar.org/e7f5/dd2277375eb335afea49583dd9ee3b057059.pdf

[6]. Goyal, S., & Parashar, A. (2016). A Proposed Solution to Knapsack Problem Using Branch & Bound Technique. (IJIRMF, Ed.) *IJIRMF, 2*(7). Retrieved Nov 20, 2019, from https://www.academia.edu/27758590/A_Proposed_Solution_to_Knapsack_Problem_Using_Branch_and_Bound_Technique_Somya_Goyal_and_Anubha_Parashar?auto=download

[7]. Haddir, B., Khemakhem, M., Hannafi, s., & Wilbaut, c. (2016, June 6). A hybrid quantum particle swarm optimization for the Multidimensional Knapsack Problem. Retrieved October 2, 2019, from https://www.sciencedirect.com/science/article/abs/pii/S0952197616300963

[8]. John, S. (2016, March 29). *What is Brute Force algorithm*. Retrieved SEPT 25, 2019, from Simplicable: https://simplicable.com/cite/brute-force

[9]. Kulkarni, A. J., & Shabir, H. (2016, June 14). Solving 0–1 Knapsack Problem using Cohort Intelligence. *Int. J. Mach. Learn. & Cyber*. doi:DOI 10.1007/s13042-014-0272-y

[10]. Meng, T., & Ke-pan, Q. (2017, January). An improved fruit fly optimization algorithm for solving the multidimensional knapsack problem. *Appled Soft Computing, 50*, 79-93. doi:https://doi.org/10.1016/j.asoc.2016.11.023

[11]. Peasah, O. K., Amponsah, S., & Asamoah, D. (2011, April). Knapsack problem: A case study of garden city radio (GCR), Kumasi, Ghana. *African Journal of Mathematics and Computer Science Research, 4*(4), 170-176. Retrieved 2019, from http://www.academicjournals.org/AJMCSR

[12]. Pushpa, S. .., Mrunal, T. V., & Suhas, C. (2016). A Study of Performance Analysis on Knapsack Problem. *International Journal of Computer Applications, 975*. Retrieved August 2, 2019, from https://pdfs.semanticscholar.org/f5bc/0cba225a9051c31dd918641e23d678cee64f.pdf

[13]. Sojeva, D. (n.d.). *P versus NP problem.* Edinburgh. Retrieved from https://www.academia.edu/6675083/P_versus_NP_problem_1_P_versus_NP_problem

[14]. Study.com. (2019). *What is an Algorithm in Programming? - Definition, Examples & Analysis*. Retrieved from Study.com: https://study.com/academy/lesson/what-is-an-algorithm-in-programming-definition-examples-analysis.html