IOSR Journal of Mathematics (IOSR-JM)
e-ISSN: 2278-5728, p-ISSN: 2319-765X. Volume 16, Issue 1 Ser. II (Jan – Feb 2020), PP 43-52
www.iosrjournals.org

Comparing the Performance of Cplex and Dynamic Programming on 0/1 Knapsack Model

[1]Alabi Taiye John [2]Olayemi Michael Sunday
[1]&[2]Kogi State Polytechnic,
School of Applied Sciences, department of Mathematics and Statistics, lokoja, Nigeria.

Abstract: *The 0/1 knapsack problem is an example of the combinatorial optimization problem which is to maximize the value of the objects in the knapsack without passing its capacity, with the aim of obtaining the best solution among other solutions. Knapsacks problems appear in practical-world decision-making processes in a wide variety of fields. There are various ways to solve the knapsack problem. In this work, dynamic programming and branch and bound were presented to solve the knapsack problem, along with the analysis of its efficiency, effectiveness, accuracy, and time execution. The data were analysed with the help of a programming language MATLAB and general purpose mixed integer programming solver CPLEX was used for the analysis. The dynamic programming suffers the best execution of time. The two methods dynamic programming and branch and bound has the same optimal solution in term of accuracy which means they are both effective for the selection of items without exceeding its capacity. In other word, our best solution values match the optimal values obtained by the CPLEX mixed integer solver, except the fact that the time required for the dynamic problem is faster than that of the CPLEX mixed integer solver.*

Keywords: *Knapsack; CPLEX; Dynamic Programming; Optimization*

Date of Submission: 31-12-2019 Date of Acceptance: 15-01-2020

I. Introduction

The 0/1 knapsack problem is an example of the combinatorial optimization problem which is to maximize the value of the objects in the knapsack without passing its capacity, with the aim of obtaining the best solution among other solutions. It is a real integer problem with single main constraints. In the past studies, it was clearly showed that knapsack problem was a wild and comprehensive problem and one of the most required and demanding in our society at large. According to Yang (2016) the knapsack name was derived from the difficulty faced by a hiker packing a backpack. The hiker must choose the objects that were of most valuable collection to acquired, subject to a volume or weight limit and the size of the pack

In knapsack problem, several knapsacks are put into consideration. It is arranged in such a way that every member of the item has its own weight, value and the number of item to be included in the item is determine in a way that the total sum of the weights is more than its capacity Babaioff et. al (2007) The most common problem is solving the 0/1 knapsack problem, which limits the number x_i of copies of each kind of item to zero or one. Given a set of n^{th} items numbered from 1 up to n, each with a weights w_i and value v_i along with a maximum capacity of W.

Maximize

$$\sum_{i=1}^{n} v_i x_i \qquad 1.1$$

Subject to the constraint

$$\sum_{i=1}^{n} w_i x_i \leq V \qquad 1.2$$

And

$$0 \leq X_i \leq Z_i$$

Levitin (2003) "If one or more of the Z_i is infinite, the knapsack problem is unbounded; else, the knapsack problem is unbounded". The bounded knapsack problem can either be multiple constraint knapsack problem or 0/1 knapsack problems.

Olayemi, (2017) Describe this as "Although good algorithms have been designed for solving simple data instances of the knapsack problem, little attention has been paid to the solution hard data instances. Such instances occur when transforming general integer programming problem to the knapsack problem, and are of great real life situation and theoretical interest."

The objectives of this research work is to have a comparative analysis study of the Dynamic Programming and Branch Bound (CPLEX).

II. Literature Review

The study on knapsack model is broad. This section gives an overview of the Knapsack problem. The different methodologies in Knapsack problem such as dynamic programming and branch and bound are discussed.

Multidimensional Knapsack Problem (MDKP)

The multidimensional 0/1 knapsack problem using Dynamic programming can be explained as follows: Given a Knapsack with m-dimensions, with the capacity of the i[th] dimension being b_i, i= 1, 2,….,m. There are n different items and let W_j be the number of copies items j; j = 1; 2,., n: The j[th] item requires w_{ij} units of the i[th] dimension of the knapsack. The reward of including a single copy of item j in the knapsack v_j The purpose is to maximize the total units of the selected items. Therefore, the problem can be formulated as an integer program as seen below:

Maximize

$\sum_{i=1}^{n} v_j$ 2.1

Subject to the constraint

$\sum_{i=1}^{n} w_{ij} \leq b_i$ 2.2

$i = 1,2, \dots, m$

$x_j \leq W_j \; and \; x_j \geq 0 \; and \; integer \; j = 1,2, \dots, n$ 2.3

When the decision variables ($W_j = 1 \; and \; x_j \in 0,1, j = 1,2, \dots., n$), the problem above is known and referred to as 0/1 multidimensional knapsack problem, but when it is unrestricted to $\{0,1\}$, it is then the general multidimensional knapsack problem. Equation (2.1) computes the profit to select subset items with maximum total profit. Equation (2.2) ensures that the Knapsack constraint is enforced where by the selected subset does not exceed the capacity. The Equation (2.3) meets the binary selection for criteria. According to Levitin (2000) multidimensional knapsack problem is an NP-hard. A comprehensive research has been carried out on 0/1 multidimensional Knapsack problem, but solution techniques for general multidimensional knapsack problem are few. As per some school of thought, "a problem is NP-hard if an algorithm in solving it can be express into one for solving any NP-problem (nondeterministic polynomial time) problem". NP-hard, therefore, means "at least Table [i, 0] = 0 for $i \geq 0$ as hard as any NP-problem," although it might, in fact, be harder. Examples of NP-complete problems include the Hamiltonian cycle and traveling salesman problems. A problem is said to be "NP-hard if the existence of polynomial time solution for it implies the existence of polynomial-time solution for every problem in NP. An NP-hard problem is NP-complete if it also belongs to the class NP. The first NP-complete problem was discovered by S. Cook in 1971." Knapsack problem looks simple but it is a very hard problem, it is not known to have efficient solution. So it might give you something which is not quite optimum but good enough to practice and that is why is called NP-hard".

Akçay et. al (2007) says, In general, multidimensional Knapsack problem has been broadly applied to model real life problems. Such as Inventory allocation in an assemble-to-order system (Akcay and Xu, 2004), Capital budgeting (Lu, Chiu, and Cox, 1999), Combinatorial Auctions (dc Vries and Vohra, 2003), Stock cutting (Caprara et al.,2000) among others. The family of Knapsack problems all need a subset of some given items to be selected such that the corresponding cost sum is maximized without passing the capacity of the Knapsack(s). Different types of Knapsack problem occur, depending on the distribution of the items and backpacks: in 0/1 Knapsack problem, each item may be chosen at most once, while in the Bounded knapsack problem, we have a limited amount of each item type. The multidimensional knapsack problem happens when items are expected to be chosen from classes of disjoint items and, if several knapsacks are to be filled simultaneously, we get the multidimensional knapsack problem. Multi-Constrained knapsack problem which is the most general integer programming (IP) with positive integer coefficients. All knapsack problems belong to the family of NP-hard problems, which mean that it is very unlikely that polynomial algorithm can ever be devised for these problems.

Related Work

Knapsack problems have been intensively studied over the decades. Knapsack techniques were modified to enhance performance and new techniques are proposed. Munapo (2008) enhanced the performance of Branch and bound. The author achieved this, by generating and adding function and single constraint to Knapsack model. The Branch and the bound is then applied and the total numbers of sub problems were reduced.

Researchers have applied Dynamic programming and Branch and Bound to solve 0/1 Knapsack problems. Different Knapsack methodologies have been compared in other to evaluate performance using specific metrics.

Pushpa et. al (2016) carried out an empirical experiment to evaluate the performance of Greedy algorithm, Dynamic programming, Branch and Bound techniques of Knapsack problem. The experiment

showed that Dynamic programming is most efficient for Knapsack with small capacities. It was also observed that Dynamic programming utilized very high memory resources. Greedy algorithm did not give an optimal solution. However, Recursive Branch and Bound was the most simple and efficient irrespective of the circumstances.

Shaheen and Sleit (2016). compared Greedy, Dynamic programming, Branch and Bound and Genetic algorithms based on time and accuracy. The algorithms were implemented using C++. The data size used for the experiment ranged from 100,000 to 600,000. Dynamic programming and Branch and Bound Outperformed Genetic and Greedy algorithm in terms of accuracy of values. Branch and bound had the worst execution time followed by Dynamic programming, then Greedy algorithm and Genetic respectively.

Hristakeva and Shrestha (2005) compared several algorithms such as brute force, dynamic programming, memory functions, branch and bound, greedy, and genetic algorithms based on complexities of the algorithms and the execution of time. The algorithms were implemented. The data size was of two categories one with constant capacity for all the population sizes increases and the other was a constant item as 500 with increases capacity. Dynamic and genetic algorithms outperformed the other approaches, but dynamic was being considered as the best among the two methods because of the complexity and rigorous situation involve in understanding and written genetic algorithms and codes.

Yanghong et. al (2017) "carried out an empirical experiment to evaluate solving 0/1 knapsack problem by a novel binary monarch butterfly optimization (BMBO) method. Two tuples, consisting of real-valued vector and binary vector were used to represent the monarch butterfly individuals in binary monarch butterfly optimization (BMBO). It was discovered that monarch butterfly optimization works directly on real-valued vectors, while binary vectors represent solutions. For a better result to be obtained three kinds of individual allocation schemes were tested. In order the revised the infeasible solutions and optimizing the feasible solutions, based on greedy strategy, a novel repaired operator was employed. Complete numerical experimentations on three types of 0/1 knapsack problem instances were carried out, the binary monarch butterfly optimization (BMBO) showed a better result in team of accuracy, convergence capacity and stability in solving 0/1 knapsack problem".

Stephen and Evans (2016) compared Greedy, Dynamic programming and branch and bound using memory request and programming effort required to implement the three algorithms as metrics. The study algorithm showed that the programming effort required for implementing each algorithm varies. The comparative study showed that memory utilizedby Dynamic programming increased exponentially, branch and bound increased slightly with increased while Greedy had optimal memory utilization.

Vikas and Shivali (2014) carried out an experiment to solve the non-fractional knapsack problem in stochastic, tournament selections and the likes. The results were compared to greedy and dynamic programming techniques of 0/1 knapsack problem. The authors varied the selection function and results were captured. The result showed that dynamic programming technique required more time and less efficient for selection problem.

Sajjan et. al (2014) carried out an experiment to research on the on a new approach to solve knapsack problem. In their argument a new approach for solving knapsack problem and testing the performance was presented. The algorithm used was originated from a common continued fraction approach and tested in polynomial time of the input length. It was discovered that the result obtained seemed not to be working perfect in term of generalization of the method.

Truong et. al (2013) researched on chemical reaction optimization with greedy (CROG) strategy for the 0/1 knapsack problem. The authors argued that, 0/1 knapsack problem was an NP- hard problem that takes a crucial parts in computing the theory and practical situation of life's. The authors say that the technique was a new optimization frame work. Truong et. Al (2013) also proposed a new chemical reaction optimization with greedy strategy algorithm. The results showed that the proposed algorithm outperformed when compared with Ant colony algorithm (ACO), genetic algorithm (GA), and quantum-inspired evolutionary algorithm (QEA) for all proposed test cases. The authors say new approach solution was only better for a short time and they intends to have full studies on the parameter values to improve the performances of the algorithm and implementation in future.

Rahajoe and Winarko (2012) carried out an experiment on Optimal Solution of MinMax 0/1 Knapsack Problem Using Dynamic Programming. The result of the experiment showed that MinMax 0/1 knapsack problem can be solved using dynamic programming in such a way that without passing the maximum capacity while a minimum limit is required the total value of items is optimal (in the case of minimal). The authors also said that MinMax 0/1 knapsack problem can also be applicable to the problem of loading commodities into the containerthe capacity requirement of a containers is met as well as the total weights is minimum without passing the given capacity of the container.

Da Silva et. al (2008) carried out an experiment on the concept of core problem in bi-criteria {0, 1}-knapsack problem. However, Da Silva et. al (2008) "it was discovered in the research that this augmentation is not of little value, since many cores can be defined for each productive solution. The experiments were conducted on five types of instances revealed that the attributes of the single criterion case also hold true for the

bi-criteria instances: they both have small size cores that increase slightly with the size of the problem. The results showed that even in the worst cases of bi-criteria core size, very few variables of the continuous solution were changed".

Rong and Figueira (2013) carried out an experiment on a reduction dynamic programming algorithm for the bi-objective integer knapsack problem. The authors developed the algorithm in order to reduce the problem built after applying variables fixing techniques based on the core concept. "A new backward state reduction RDP algorithm for bi-objective integer knapsack problem were presented based on based on the construction of a mixed network containing items with different upper bound". The result showed that, the proposed RDP algorithm has a better response to the problem reduction as compared with the benchmark algorithm. Also the RDP algorithm showed significant solution time advantage over the benchmark for both the original and the reduced problem.

III. Methodologies And Algorithm

Introduction

As earlier discussed there are several approaches in which can be used to solve 0/1 knapsack problems. However, for this work dynamic programming and branch and bound algorithms were used to solve the 0/1 Knapsack problem. This chapter presents how the testing of the methodologies was carried out.

Environment Setup and Hardware

The program and data were implemented on a window 7 computers with 8 GBRAM and intel(R) core (TM) i5 2005 CPU 3.30GHz processor.

Data Set

In this work, self-integer data were randomly generated through MATLAB. They will be used as weights and values of some certain items, this aimed at maximizing the values of those items. The data were analysed with the help of a programming language MATLAB and general purpose mixed integer programming solver CPLEX was used for the analysis.

Dynamic programming algorithm

The Dynamic programming is an algorithm for solving problems that are classify as optimization problems with the aim of calculating the solutions to sub-problems once and then store it results in a table to be able to recall and used in the future.

1) Describe the arrangement of an optimal solution by extract the problem into small problems, and look about a connection between the arrangement of the optimal solution of the real problem and the solutions of the smaller problems.

2) Define the optimal solution Recursively by express the solution of the first (real) problem in terms of optimal solutions for smaller problems.

3) Calculate the value of an optimal solution in a bottom-up approach by using a table.

4) Construct an optimal solution from computed information

Dynamic Programing pseudo code for solving knapsack problem

Input:

1. Array of Value (v).
2. Array of Weights (w).
3. Number of items (n)
4. Capacity (W)

```
Dynamic Programming (w.v.W){
for i = 0 to W do
B[0,i] = 0
end for
for i = 1 to n do
for wk = 0 to W do
if v[i] ≤ wk then
B[i,wk] = max (B[i-1,wk],B[i-1,wk-v[i]] +c[i])
else
B[i,wk] = B[i-1,wk]
end if
end for
end for
}
Return Max Value
```

Descriptions of cplexbilp for solving 0/1 knapsack problem (maximization problem)

f= - double column vector for the objective function, that is the values
Aeq = double matrix for linear equality constraints
beq = double column vector for linear equality constraints
Aineq = double matrix for linear inequality constraints
bineq = double column vector for linear inequality constraints that is capacity
x = cplexbilp(f,Aineq,bineq,Aeq,beq);

IV. Analysis And Discussion

Analysis of Result

For testing two different algorithms, files were generated with different sizes where each record consists of a pair of randomly generated integers representing the value and weight of each item. In this testing, different capacity was chosen for each of the population depending on the size of the population as the population size increases so also the capacity of each population size increases, but e of the capacity is more than the sum of the weight in each of the population sizes. Time execution: The execution time metric measures to what extent do the algorithm take to finish. To obtained the required time to be estimated, time complexity is the worst to solve the 0/1 knapsack problem as a function of input data size. The time execution played a large part in the enhancement of the performance of the system.

Summary of the output of dynamic programming using MATLAB

Table 1

Population size	Capacity	Maximum values & weights found	Items chosen	Time (seconds)
10	173	462 & 132	2, 3, 8, 9, 10	0.037495
50	102	409 & 101	1, 4, 14, 21, 25, 26, 41	0.050923
100	120	745 & 119	3, 7, 10, 19, 45, 56, 58, 60, 63, 80, 92	0.036539
200	250	1519 & 249	10, 29, 39, 41, 57, 61, 78, 79, 85, 97, 125, 141, 145, 148, 155, 158, 159, 160, 161, 167, 171, 173, 184, 199	0.057846
500	600	4084 & 600	7, 19, 20, 24, 30, 43, 48, 55, 57, 59, 64, 78, 86, 90, 106, 120, 122, 126, 139, 140, 143, 144, 148, 152, 165, 166, 177, 191, 206, 208, 231, 241, 259, 265, 276, 278, 308, 329, 368, 382, 387, 393, 398, 408, 412, 421, 430, 431, 433, 434, 436, 453, 461, 471, 472, 480, 494, 496, 497	0.070535
600	600	4656 & 600	9, 25, 35, 36, 43, 46, 49, 53, 61, 66, 73, 75, 78, 82, 110, 120, 126, 132, 142, 144, 156, 158, 162, 167, 173, 189, 196, 197, 198, 204, 226, 234, 240, 251, 260, 270, 272, 277, 279, 282, 305, 313, 332, 344, 348, 350, 360, 363, 371, 383, 426, 432, 447, 449, 454, 498, 503, 507, 516, 527, 528, 538, 540, 587, 595	0.092876
700	800	6135 & 800	1, 21, 23, 31, 44, 49, 68, 78, 99, 103, 106, 120, 123, 124, 136, 144, 158, 160, 179, 180, 181, 193, 219, 224, 229, 234, 241, 264, 265, 272, 273, 274, 279, 280, 283, 290, 304, 307, 342, 344, 353, 354, 356, 366, 375, 377, 386, 397, 406, 409, 413, 414, 420, 433, 438, 445, 447, 454, 460, 468, 480, 492, 496, 503, 510, 512, 516, 530, 534, 538, 542, 552, 555, 561, 567, 571, 575, 577, 581, 596, 598, 621, 649, 653, 659, 662, 668, 698, 700	0.080616
800	850	6293 & 850	5, 11, 15, 24, 25, 36, 39, 41, 43, 49, 64, 75, 78, 88, 104, 118, 124, 153, 159, 183, 192, 199, 207, 231, 233, 236, 237, 240, 241, 247, 248, 251, 264, 270, 277, 279, 289, 292, 328, 330, 341, 367, 374, 379, 404, 417, 427, 430, 446, 454, 472, 480, 516, 517, 527, 534, 538, 546, 549, 558, 561, 568, 575, 580, 584, 592, 596, 601, 610, 611, 619, 620, 629, 663, 668, 677, 679, 686, 687, 688, 694, 695, 698, 701, 709, 711, 714, 733, 761, 765, 768, 769, 775, 779, 791,795	0.077180
900	1000	7289 & 1000	13, 14, 17, 26, 50, 63, 73, 77, 80, 82, 84, 91, 96, 98, 101, 109, 121, 128, 131, 144, 146, 149, 178, 199, 203, 206, 217, 220, 224, 244, 258, 260, 267, 268, 269, 271, 275, 280, 293, 319, 324, 329, 334, 341, 354, 362, 364, 365, 372, 379, 380, 383, 386, 390, 433, 442, 444, 466, 477, 481, 486, 497, 506, 509, 510, 513, 514, 515, 520, 531, 533, 545, 547, 565, 580, 600, 612, 616, 622, 630, 633,634, 638, 642, 652, 655, 656, 662, 667, 681, 696, 698, 723, 749, 753, 759, 798, 800, 819, 829, 849, 856,	0.088933

			872, 877, 888, 890, 896, 898	
1000	1200	9232 & 1200	6, 22, 30, 34, 35, 40, 55, 68, 69, 92, 96, 99, 101, 105, 112, 123, 127, 146, 148, 152, 170, 174, 175, 190, 216, 224, 241, 244, 261, 265, 271, 281, 294, 296, 301, 303, 304, 306, 310, 319, 321, 325, 333, 336, 357, 361, 376, 396, 402, 408, 409, 440, 448, 452, 460, 461, 475, 488, 506, 518, 519, 520, 532, 545, 556, 578, 584, 592, 594, 600, 609, 635, 636, 643, 648, 651, 653, 661, 672, 673, 678, 682, 692, 699, 720, 721, 723, 732, 739, 742, 744, 752, 756, 758, 762, 763, 772, 773, 784, 789, 790, 798, 805, 816, 824, 834, 841, 860, 865, 869, 870, 871, 872, 877, 880, 881, 882, 884, 893, 894, 905, 932, 948, 951, 956, 969, 971, 999	0.107365

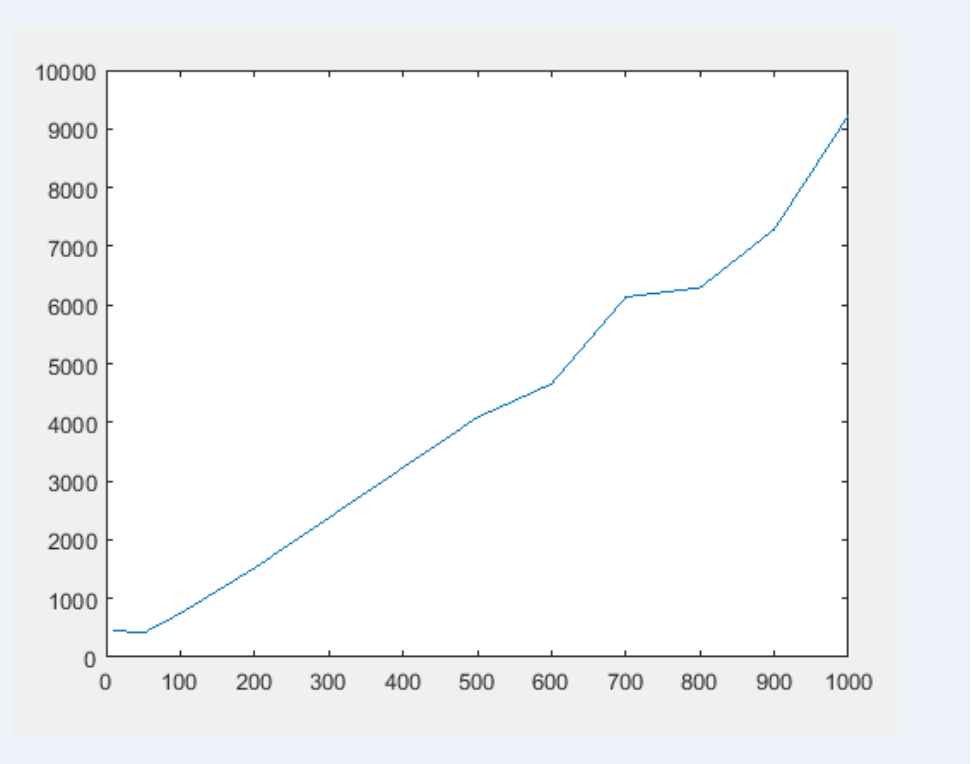
Summary of the output of branch and bound using CPLEXBILP in MATLAB

Table 2.

Population size	Capacity	Maximum values & weights found	Items chosen	Time (seconds)	Number of item generated
10	173	462 & 132	2, 3, 8, 9, 10	0.593699	5
50	102	409 & 101	1, 4, 14, 21, 25, 26, 41	0.468606	7
100	120	745 & 119	3, 7, 10, 19, 45, 56, 58, 60, 63, 80, 92	0.500475	11
200	250	1519 & 249	10, 29, 39, 41, 57, 61, 78, 79, 85, 97, 125, 141, 145, 148, 155, 158, 159, 160, 161, 167, 171, 173, 184, 199	0.394070	24
500	600	4084 & 600	7, 19, 20, 24, 30, 43, 48, 55, 57, 59, 64, 78, 86, 90, 106, 120, 122, 126, 139, 140, 143, 144, 148, 152, 165, 166, 177, 191, 206, 208, 231, 241, 259, 265, 276, 278, 308, 329, 368, 382, 387, 393, 398, 408, 412, 421, 430, 431, 433, 434, 436, 453, 461, 471, 472, 480, 494, 496, 497	0.157893	59
600	600	4656 & 600	9, 25, 35, 36, 43, 46, 49, 53, 61, 66, 73, 75, 78, 82, 110, 120, 126, 132, 142, 144, 156, 158, 162, 167, 173, 189, 196, 197, 198, 204, 226, 234, 240, 251, 260, 270, 272, 277, 279, 282, 305, 313, 332, 344, 348, 350, 360, 363, 371, 383, 426, 432, 447, 449, 454, 498, 503, 507, 516, 527, 528, 538, 540, 587, 595	0.313513	63
700	800	6135 & 800	1, 21, 23, 31, 44, 49, 68, 78, 99, 103, 106, 120, 123, 124, 136, 144, 158, 160, 179, 180, 181, 193, 219, 224, 229, 234, 241, 264, 265, 272, 273, 274, 279, 280, 283, 290, 304, 307, 342, 344, 353, 354, 356, 366, 375, 377, 386, 397, 406, 409, 413, 414, 420, 433, 438, 445, 447, 454, 460, 468, 480, 492, 496, 503, 510, 512, 516, 530, 534, 538, 542, 552, 555, 561, 567, 571, 575, 577, 581, 596, 598, 621, 649, 653, 659, 662, 668, 698, 700	0.569219	89
800	850	6293 & 850	5, 11, 15, 24, 25, 36, 39, 41, 43, 49, 64, 75, 78, 88, 104, 118, 124, 153, 159, 183, 192, 199, 207, 231, 233, 236, 237, 240, 241, 247, 248, 251, 264, 270, 277, 279, 289, 292, 328, 330, 341, 367, 374, 379, 404, 417, 427, 430, 446, 454, 472, 480, 516, 517, 527, 534, 538, 546, 549, 558, 561, 568, 575, 580, 584, 592, 596, 601, 610, 611, 619, 620, 629, 663, 668, 677, 679, 686, 687, 688, 694, 695, 698, 701, 709, 711, 714, 733, 761, 765, 768, 769, 775, 779, 791,795	0.498768	96
900	1000	7289 & 1000	13, 14, 17, 26, 50, 63, 73, 77, 80, 82, 84, 91, 96, 98, 101, 109, 121, 128, 131, 144, 146, 149, 178, 199, 203, 206, 217, 220, 224, 244, 258, 260, 267, 268, 269, 271, 275, 280, 293, 319, 324, 329, 334, 341, 354, 362, 364, 365, 372, 379, 380, 383, 386, 390, 433, 442, 444, 466, 477, 481, 486, 497, 506, 509, 510, 513, 514, 515, 520, 531, 533, 545, 547, 565, 580, 600, 612, 616, 622, 630, 633,634, 638, 642, 652, 655, 656, 662, 667, 681, 696, 698, 723, 749, 753, 759, 798, 800, 819, 829, 849, 856, 872, 877, 888, 890, 896, 898	0.535026	108
1000	1200	9232 & 1200	6, 22, 30, 34, 35, 40, 55, 68, 69, 92, 96, 99, 101, 105, 112, 123, 127, 146, 148, 152, 170, 174, 175, 190, 216, 224, 241, 244, 261, 265, 271, 281, 294, 296, 301, 303, 304, 306, 310, 319, 321, 325, 333, 336, 357, 361, 376, 396, 402, 408, 409, 440, 448, 452, 460, 461, 475, 488, 506, 518, 519, 520, 532, 545, 556, 578, 584, 592, 594, 600, 609, 635, 636, 643, 648, 651, 653, 661, 672, 673, 678, 682, 692, 699, 720, 721, 723, 732, 739, 742, 744, 752, 756, 758, 762, 763, 772, 773, 784, 789, 790, 798, 805, 816, 824, 834, 841, 860, 865, 869, 870, 871, 872, 877, 880, 881, 882, 884, 893, 894, 905, 932, 948, 951, 956, 969, 971, 999	0.281075	128

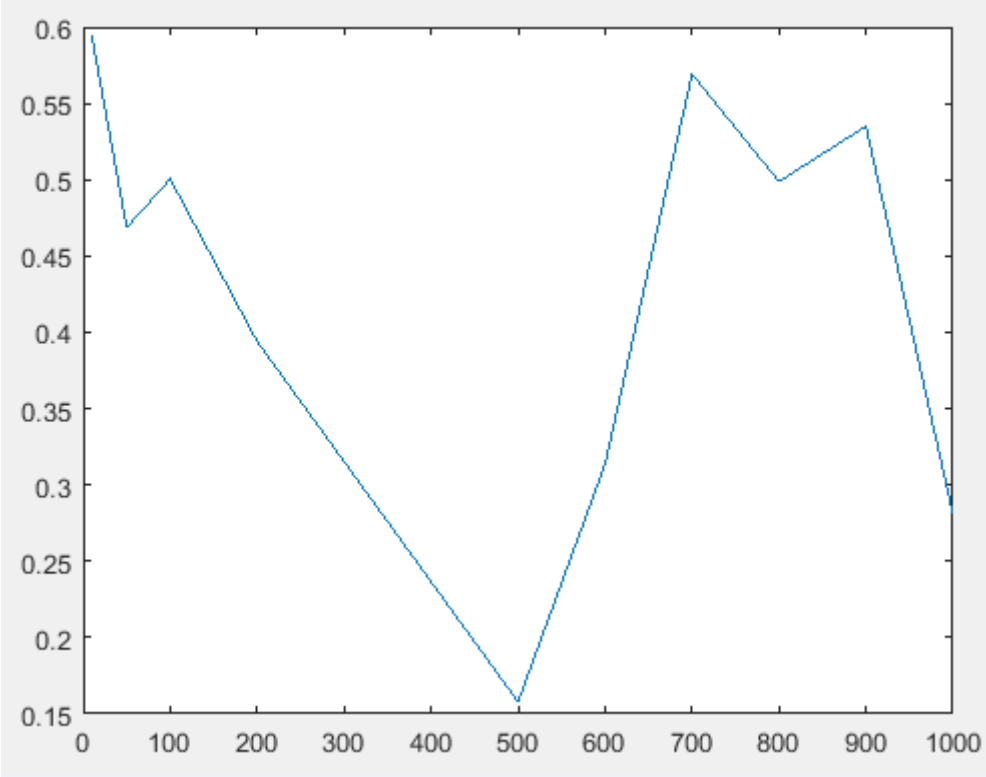
The plot of the population size and maximum values obtained for the branch and bound algorithm.

Figure 1.

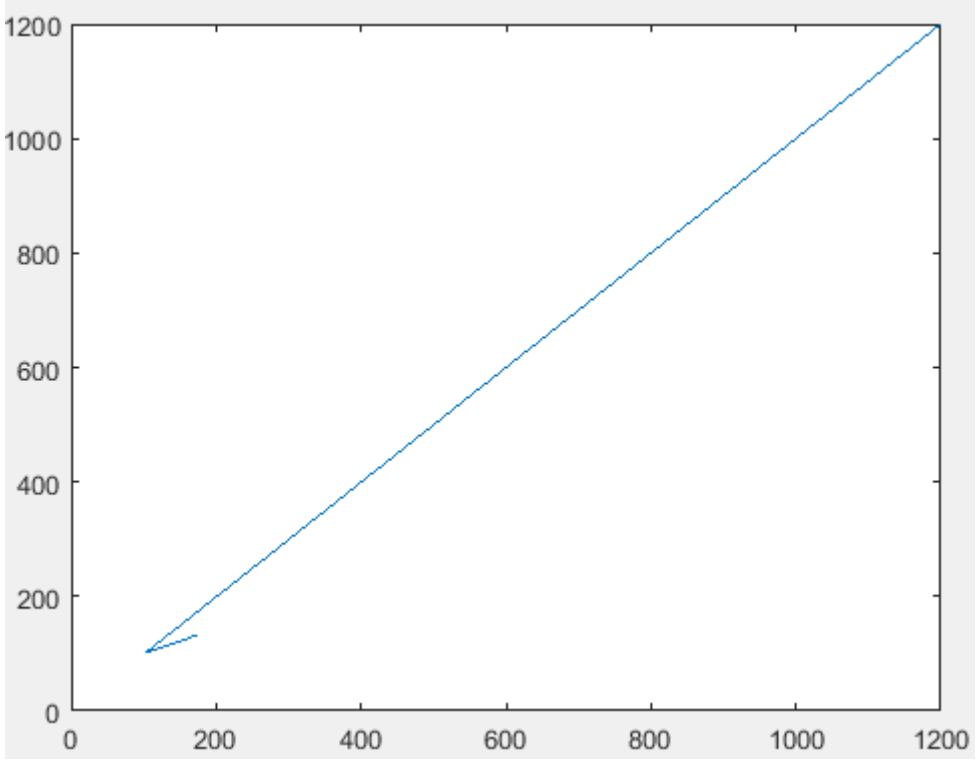


The plot of the population size and time (seconds) obtained for the branch and bound algorithm.

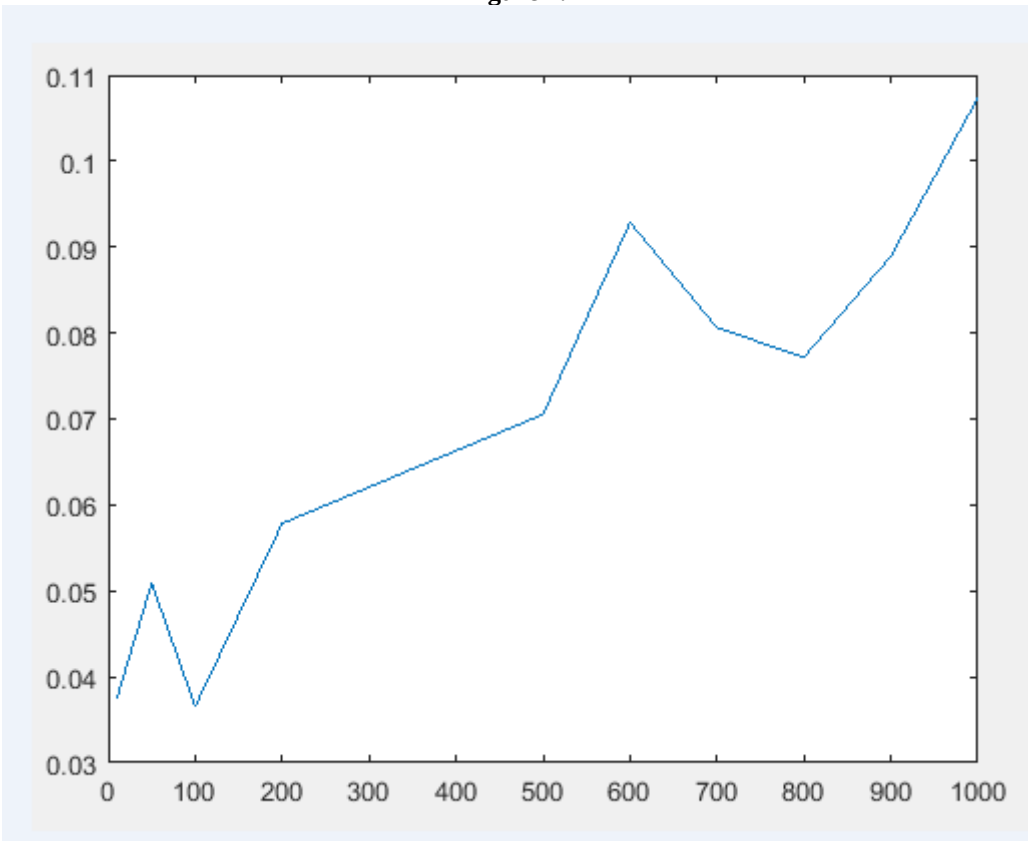
Figure 2.



The plot of the capacity and maximum weights obtained for the branch and bound algorithm.
Figure 3.



The plot of the population size and time (seconds) obtained for the dynamic programming algorithm.
Figure 4.



V. Discussion of Result

The analysis includes the following parameters; execution time and their efficiency to get the maximum value into the knapsack.

The aim of any algorithm solving knapsack problem is to present high-yielding solution in the minimum possible time.

For branch and bound (CPLEX), as seen in the table 2 above the highest population size randomly generated is 1000 with the highest optimum solution of 9232 with full capacity of 1200 which has 128 selected items and it run only in 0.281075 seconds. The running time lies between 0.157893 seconds and 0.593699 seconds. From table 1 above, it shows clearly that the size of the population does not determine how high or low the time will be. As shown in table 2 above where the least population size 10 is having the highest time as 0.593699 seconds where the highest population size 1000 with time as 0.281075 seconds.

In figure 1 above It clearly shows that as the population size is increases show also the there is more chance to obtained higher optimal solution in as much as the capacity for the next population size increasing as well. In figure 2 It shows that the timing is fluctuating against the population size, but highest time it takes to run code for the CPLEX mixed integer solver is 0.593699 seconds which is less than a minute. That shows how good and efficient the code is and that will generate an effective output as seen in the summary table in table 2. If the capacity of the knapsack is less than the size of population, as the capacity is increases show also the maximum weights will be increases. The maximum weight will never exceed the capacity but they could be equal, and then it will end up with a straight line (that is, the higher the capacity, the higher the maximum weight obtained) as seen in figure 2.

From the output of the dynamic programming it is clearly shows that the optimum solution for all the population sizes are the same as that of the branch and bound CPLEX mixed integer solver the number of the selected item are the same as well. But the highest time to run the data of 1000 population size is 0.107305 in seconds; this implies that it is a good result. Provided the capacity of the knapsack is greater than size of the population, the dynamic programming and branch and bound number of operations will be higher as well. Our best solution values match the optimal values obtained by the CPLEX mixed integer solver, except the fact that the time required for the dynamic problem is faster than that of the CPLEX mixed integer solver

VI. Conclusion

The dynamic programming and CPLEX have been presented. The comparisons of the analysis performed and conducted have been presented, and compared to experiment result obtained from applying these algorithms on 0/1 knapsack problem.

Table 3: CPLEX V.s Dynamic

Population size	CPLEX	Dynamic programming
10	462	462
50	409	609
100	745	745
200	1519	1519
500	4084	4084
600	4656	4656
700	6135	6135
800	6293	6293
900	7289	7289
1000	9232	9232

The comparative study of the branch and bound (CPLEX mixed integer programming solver) and dynamic programming shows while the complexities of these algorithms are known. The results demonstrate the effectiveness of the algorithms, in terms of execution of time.

The two method are effective and efficient to use, but the nature of the problem make one suitable than the other in the aspect of time and the complexities in understanding the algorithm. The best approximation approach around time execution is dynamic programming, although the optimal solution obtained for the two algorithms are the same but the time required for dynamic to execute the data is less than that of branch and bound (CPLEX mixed integer programming solver). In other word branch and bound (CPLEX mixed integer programming solver) suffered worst execution of time while dynamic programming suffers the best execution of time. However, one may choose branch and bound (CPLEX mixed integer programming solver) over dynamic programming algorithms in other circumstance, because it is easy and straightforward to code. In contrast, dynamic programming algorithms require a lot more time in term of understanding the concepts of the pattern and in term of programming effort.

For future work, we intend to implement more 0/1 knapsack problems techniques. Apply the techniques to solve mathematical or real life problems. Comparison will be made to discover the most suitable technique for the selected problem.

Acknowledgement

We wish to acknowledge TETFUND Nigeria and management of Kogi State Polytechnic, Lokoja for their contributions toward this work.

References

[1]. Akçay, Y., Li, H., & Xu, S. H. (2007). Greedy algorithm for the general multidimensional knapsack problem. Annals of Operations Research, 150(1), 17-29.
[2]. Babaioff, M., Immorlica, N., Kempe, D., & Kleinberg, R. (2007). A knapsack secretary problem with applications. Approximation, randomization, and combinatorial optimization. Algorithms and techniques, 16-28.
[3]. Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G., & Stewart, W. R. (1995). Designing and reporting on computational experiments with heuristic methods. Journal of heuristics, 1(1), 9-32.
[4]. Chu, P. C., & Beasley, J. E. (1997). A genetic algorithm for the generalised assignment problem. Computers & Operations Research, 24(1), 17-23.
[5]. Da Silva, C. G., Clímaco, J., & Figueira, J. R. (2008). Core problems in bi-criteria {0, 1}-knapsack problems. Computers & Operations Research, 35(7), 2292-2306.
[6]. Feng, Y., Wang, G. G., Deb, S., Lu, M., & Zhao, X. J. (2017). Solving 0–1 knapsack problem by a novel binary monarch butterfly optimization. Neural computing and applications, 28(7), 1619-1634.
[7]. Forin, A. (1988). Design, implementation, and performance evaluation of a distributed shared memory server for Mach.
[8]. Garey, M. R., & Johnson, D. S. (1979). Computers and intractability, vol. 174.
[9]. Hanafi, S., & Freville, A. (1998). An efficient tabu search approach for the 0–1 multidimensional knapsack problem. European Journal of Operational Research, 106(2-3), 659-675.
[10]. Hristakeva, M., & Shrestha, D. (2005, April). Different approaches to solve the 0/1 knapsack problem. In The Midwest Instruction and Computing Symposium.
[11]. Kolesar, P. J. (1967). A branch and bound algorithm for the knapsack problem. Management science, 13(9), 723-735.
[12]. Lee, J. (2004). A first course in combinatorial optimization (Vol. 36). Cambridge University Press.
[13]. Levitin, A. (2000). Design and analysis of algorithms reconsidered. ACM SIGCSE Bulletin, 32(1), 16-20.
[14]. Monapo, E. (2008). The efficiency enhanced branch and bound algorithm for the Knapsack model. Adv. Appl. Math. Anal, 3(1), 81-89.
[15]. Olayemi, M.S. (2017). MA902 Research Methods submitted to the department of Mathematical Sciences, University of Essex.
[16]. Oppong, S. O., & Baidoo, E. Exploring Traditional Approaches for Solving 0-1 Knapsack Problem.
[17]. Pisinger, D. (1997). A minimal algorithm for the 0-1 knapsack problem. Operations Research, 45(5), 758-767.
[18]. Pushpa, S.k., Mrunal, T.V., & Suhas, C. (2016). A Study of Performance Analysis on Knapsack Problem. International Journal of Computer Applications, 0975 – 8887
[19]. Puterman, M. L. (2014). Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons.
[20]. Rahajoe, A. D., & Winarko, E. (2012). Optimal Solution Of Minmax 0/1 Knapsack Problem Using Dynamic Programming. International Journal of Informatics and Communication Technology (IJ-ICT), 2(1), 9-16.
[21]. Rong, A., & Figueira, J. R. (2013). A reduction dynamic programming algorithm for the bi-objective integer knapsack problem. European Journal of Operational Research, 231(2),299-313.
[22]. Sajjan S.P.., Roogi R., Badiger1 V., & Amaragatti S.A. (2014). New Approach to solve Knapsack Problem.
[23]. Shaheen, A., & Sleit, A. (2016). Comparing between different approaches to solve the 0/1
[24]. Knapsack problem. International Journal of Computer Science and Network Security (IJCSNS), 16(7), 1.
[25]. Thada, V., & Dhaka, S. (2014). Genetic Algorithm based Approach to Solve Non Fractional (0/1) Knapsack Optimization Problem. International Journal of Computer Applications, 100(15), 21-26.
[26]. Tian, Y., Lv, J., & Zheng, L. (2013). An algorithm of 0-1 knapsack problem based on economic model. Journal of Applied Mathematics and Physics, 1(04), 31.
[27]. Toth, P. (2000). Optimization engineering techniques for the exact solution of NP-hard combinatorial optimization problems. European journal of operational research, 125(2), 222-238.
[28]. Truong, T. K., Li, K., & Xu, Y. (2013). Chemical reaction optimization with greedy strategy for the 0–1 knapsack problem. Applied Soft Computing, 13(4), 1774-1780.
[29]. Weisstein, E. W. (2008). Floyd-Warshall Algorithm.
[30]. Yadav, V., & Singh, S. (2016) A reviewed paper on 0/1 knapsack problem with genetic algorithms. International journal of computer science and information technologies, 7 (2), 830-832.
[31]. Yang, X. (2016). Combinatorial optimization lecture note.

Alabi Taiye John.et.al. "Comparing the Performance of Cplex and Dynamic Programming on 0/1 Knapsack Model." *IOSR Journal of Mathematics (IOSR-JM)*, 16(1), (2020): pp. 43-52.