

# Dynamic Programming Algorithm using Furniture Industry 0/1 Knapsack Problem

1<sup>st</sup> Mietha Anditta  
Computer Science  
Department,  
School of Computer Science.  
Bina Nusantara University.  
Jakarta, Indonesia 11530.  
[mietha.anditta@binus.ac.id](mailto:mietha.anditta@binus.ac.id)

2<sup>nd</sup> Natashya Amartya  
Computer Science  
Department, School of  
Computer Science.  
Bina Nusantara University.  
Jakarta, Indonesia 11530.  
[mietha.anditta@binus.ac.id](mailto:mietha.anditta@binus.ac.id)

3<sup>rd</sup> Laurens Spits Warnars  
Information Systems  
Department, School of  
Information Systems.  
Bina Nusantara University.  
Jakarta, Indonesia 11530.  
[laurens.warnars@binus.ac.id](mailto:laurens.warnars@binus.ac.id)

4<sup>th</sup> Harco Leslie Hendric Spits  
Warnars  
Computer Science Department,  
BINUS Graduate Program -  
Doctor of Computer Science.  
Bina Nusantara University.  
Jakarta, Indonesia 11530.  
[spits.hendric@binus.ac.id](mailto:spits.hendric@binus.ac.id)

5<sup>th</sup> Arief Ramadhan  
PJJ Informatika.  
Telkom University.  
Bandung, Indonesia 40257.  
[ariefmadhan@telkomuniversity.ac.id](mailto:ariefmadhan@telkomuniversity.ac.id)

6<sup>th</sup> Teddy Siswanto  
Information Systems  
Department.  
Trisakti University.  
Jakarta, Indonesia 11440.  
[teddysiswanto@trisakti.ac.id](mailto:teddysiswanto@trisakti.ac.id)

7<sup>th</sup> Teddy Mantoro  
Faculty of Engineering and  
Technology.  
Sampoerna University  
Jakarta, Indonesia 12760  
[teddy.mantoro@sampoernauniversity.ac.id](mailto:teddy.mantoro@sampoernauniversity.ac.id)

8<sup>th</sup> Nurulhuda Noordin  
School of Computing Sciences  
College of Computing, Informatics  
and Mathematics.  
Universiti Teknologi MARA.  
Shah Alam, Selangor, Malaysia.  
[drnurul@uitm.edu.my](mailto:drnurul@uitm.edu.my)

**Abstract**—Dynamic programming is a fundamental algorithm that can be found in our daily lives easily. One of the dynamic programming algorithm implementations consists of solving the 0/1 knapsack problem. A 0/1 knapsack problem can be seen from industrial production cost. It is prevalent that a production cost has to be as efficient as possible, but the expectation is to get the proceeds of the products higher. Thus, the dynamic programming algorithm can be implemented to solve the diverse knapsack problem, one of which is the 0/1 knapsack problem, which would be the main focus of this paper. The implementation was implemented using C language. This paper was created as an early implementation algorithm using a Dynamic program algorithm applied to an Automatic Identification System (AIS) dataset.

**Keywords**—Dynamic programming, 0/1 knapsack problem, Optimization methods, Optimization Problem, Optimization.

## I. INTRODUCTION

Optimisation is the process of achieving the best possible solution under given circumstances. Its goal is none other than both to maximise the benefits and to minimise the effort. The optimisation field includes many different fields and algorithms. The field base can be classified in many other areas; for example, it can be divided into two categories, which depend on whether the variables are discrete or continuous. Besides, one optimisation problem can be solved with multiple approaches, which is why it includes various algorithms. One of the well-known optimisation problems is a knapsack.

The knapsack problem is a non-deterministic polynomial-time (NP) complex problem since the algorithm can be translated into one to solve other NP problems. A knapsack problem is finding the number of items that could be included within the threshold  $W$  and the maximum value of the utility score if given a set of items with their weights and utility scores. One condition is that each item can be either selected or not and

can't be fractionally selected. This is the difference between a 0/1 knapsack and a fractional knapsack. In 0/1 knapsack, an item counts as a whole, which is different from a fractional knapsack in that a fractional part of each item can also be selected. The 0/1 knapsack problem can be seen in choosing if a phone should be fixed, while the fractional knapsack problem is more like considering how much of the one bottle of tea should be selected.

The mathematical model of the 0/1 knapsack problem may be instances as below, where  $x_i$  represents the number of  $i$  instances from a set of  $n$  items with  $W_i$  as weight and  $v_i$  as value.

$$\text{Maximize } \sum_{i=1}^n x_i v_i \quad (1)$$

$$\text{Subject to } \sum_{i=1}^n x_i w_i \leq W, \quad x_i \in \{0,1\}, \forall i \in \{1,2,3,\dots,n\} \quad (2)$$

The knapsack problem has been studied in the importance of real-world applications for centuries, and many algorithms have been suggested to solve the well-known knapsack problem [1]. Those algorithms presented over the century to solve the knapsack problem are greedy methods, dynamic programming, etc. The first appearance of the knapsack problem itself was in 1957, written in a paper by George Dantzig, son of the earlier mathematician Tobias Dantzig, who demonstrated the persistence of the knapsack problem [2].

Nowadays, knapsack problems can be easily found in society, whether on a small scale, like what to bring in a bag or even on a bigger scale, such as in production. For example, price is one of the customer's satisfaction factors, and a company should determine the appropriate price for their customer. In determining the price, they have to know about the production cost, which includes the material cost that can be assumed as a 0/1 knapsack problem. Their main goal in

calculating the material cost is finding the best materials with the least cost to determine the perfect worth for each product. This 0/1 knapsack problem can be solved by using the dynamic programming approach.

Dynamic programming is one of the mathematical optimisation methods in computer programming, and it's also one of the most used algorithms in solving the 0/1 knapsack problem. This algorithm generally computes all the sub-problems' solutions and then stores them in a table to reuse later [3]. The algorithm denotes the optimal solution structure by deriving the main problem into the smaller one and then finding the correlation of each optimal solution structure from both of it, specifying the optimal solution value recursively by expressing the main problem's solution regarding the optimal solution for the minor issue, calculate the optimal solution value by using a table with a bottom-up approach and construct the optimal solution based on the computed information[4].

Since this algorithm is excellent for optimisation problems, the rest of this paper will present the usage of dynamic programming algorithms in one of the most often encountered problems in the production field. As one of the solutions for the 0/1 knapsack problem, dynamic programming algorithms will be studied and evaluated analytically and experimentally in the matter of the total value and time[5].

## II. LITERATURE REVIEW PREVIOUS PUBLICATIONS

A search on Google Scholar (<https://scholar.google.com/>) using the keywords from the title of the proposed paper, namely "Dynamic Programming Algorithm 0/1 knapsack problem", shows there are 30,400 search results ([https://scholar.google.com/scholar?hl=en&as\\_sdt=0%2C5&as\\_vis=1&q=Dynamic+Programming+Algorithm+0%2F1+knapsack+problem&btnG=](https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&as_vis=1&q=Dynamic+Programming+Algorithm+0%2F1+knapsack+problem&btnG=)) and if limited to the last 5 years there are 11,800 search results ([https://scholar.google.com/scholar?as\\_ylo=2019&q=Dynamic+Programming+Algorithm+0/1+knapsack+problem&hl=en&as\\_sdt=0.5&as\\_vis=1](https://scholar.google.com/scholar?as_ylo=2019&q=Dynamic+Programming+Algorithm+0/1+knapsack+problem&hl=en&as_sdt=0.5&as_vis=1)). However, this search result is not valid considering that this search is based on all written content, and if the search is limited to only previous publication titles, then only 5 search results will be obtained

([https://scholar.google.com/scholar?as\\_q=Dynamic+Programming+Algorithm+knapsack+problem&as\\_epq=0+1&as\\_oq=&as\\_eq=&as\\_occt=title&as\\_sauthors=&as\\_publication=&as\\_ylo=&as\\_yhi=&hl=en&as\\_sdt=0%2C5&as\\_vis=1](https://scholar.google.com/scholar?as_q=Dynamic+Programming+Algorithm+knapsack+problem&as_epq=0+1&as_oq=&as_eq=&as_occt=title&as_sauthors=&as_publication=&as_ylo=&as_yhi=&hl=en&as_sdt=0%2C5&as_vis=1)). If you restrict the search to previous publication titles for the last 5 years there are only 2 search results ([https://scholar.google.com/scholar?as\\_ylo=2019&q=allintitle:+Dynamic+Programming+Algorithm+knapsack+problem+%220+1%22&hl=en&as\\_sdt=0.5&as\\_vis=1](https://scholar.google.com/scholar?as_ylo=2019&q=allintitle:+Dynamic+Programming+Algorithm+knapsack+problem+%220+1%22&hl=en&as_sdt=0.5&as_vis=1))

A further search on the title of the previous publication followed by using the keyword "Dynamic Programming Algorithm" shows that there are 1,770 search results ([https://scholar.google.com/scholar?q=allintitle:+Dynamic+programming+algorithm&hl=en&as\\_s](https://scholar.google.com/scholar?q=allintitle:+Dynamic+programming+algorithm&hl=en&as_s)

[dt=0.5](https://scholar.google.com/scholar?as_ylo=2019&q=allintitle:+Dynamic+programming+algorithm&hl=id&as_sdt=0.5)) and if limited to the last 5 years there are 302 search results ([https://scholar.google.com/scholar?as\\_ylo=2019&q=allintitle:+Dynamic+programming+algorithm&hl=id&as\\_sdt=0.5](https://scholar.google.com/scholar?as_ylo=2019&q=allintitle:+Dynamic+programming+algorithm&hl=id&as_sdt=0.5)). In addition, the search continues by searching for the title of the previous publication using the keyword "0/1 Knapsack Problem" then 577 search results were obtained ([https://scholar.google.com/scholar?q=allintitle:+0/1+Knapsack+Problem&hl=en&as\\_sdt=0.5&as\\_vis=1](https://scholar.google.com/scholar?q=allintitle:+0/1+Knapsack+Problem&hl=en&as_sdt=0.5&as_vis=1)). If you limit the search to previous publication titles for the last 5 years, there are only 115 search results ([https://scholar.google.com/scholar?as\\_ylo=2019&q=allintitle:+0/1+Knapsack+Problem&hl=en&as\\_sdt=0.5&as\\_vis=1](https://scholar.google.com/scholar?as_ylo=2019&q=allintitle:+0/1+Knapsack+Problem&hl=en&as_sdt=0.5&as_vis=1))

### A. Historical Preview of Dynamic Programming Algorithm

Richard Bellman, a mathematician from America and known as the founding father of dynamic programming algorithms, introduced this algorithm in 1952. Bellman has taught us the "principle of optimality," one of the most essential ideas for control and optimisation problems [6]. The key idea was to approximate the functions using the basis function approximation. This algorithm makes it able to reduce the computation process but with uncertainty cost after the computation [7]. This "principle of optimality" hypothesises that the optimal solution is computed at the beginning of the decision process before any uncertain data becomes available.

Bellman's first book, *An Introduction to The Theory of Dynamic Programming*, which the RAND Corporation published in 1953, mentions:

"We are informed that a particle is in either state 0 or 1, and we are initially given the probability  $x$  that it is in state 1. Operation A will reduce this probability to  $ax$ , where  $a$  is some positive constant less than 1. In contrast, operation L, which consists of observing the particle, will tell us its state. What is the optimal procedure if it is desired to transform the particle into state 0 in a minimum time?"

The statement above explains that a dynamic programming algorithm was developed to find a case's possible optimal/maximum solution.

### B. Definition of Dynamic Programming Algorithm

Dynamic programming algorithms are immensely studied and implemented in various aspects of human life. Dynamic programming focusing mainly on optimisation problems relies on Bellman's principle, where the optimal solution of each partial problem must have the constitution of the previous optimal solution [8]. It is mainly applied when the problem can be divided into several partial problems, where discovering the maximum efficient approach to cut insufficient materials for benefits resources or as adaptive dynamic programming [9]. To solve a problem using dynamic programming, each partial problem should be solved individually. Then, to get the optimal solution, combine all of the sub-solutions. Two critical characteristics of issues can be solved by applying a dynamic programming algorithm [2]: firstly, the overlapping subproblems. A recursive algorithm is used in a dynamic programming algorithm to solve the problem with the same method [2]—secondly, the optimal substructures. The optimal

solution for the optimisation can be obtained from the optimal solution of each subproblem.

In dynamic programming, it solves each partial or more minor problem to store the solution in a table. Its purpose is to use the solution in the future as a parameter to achieve the next possible optimal solution because it has its characteristics where the optimal solution can be achieved through deriving it from the partial problems recursively.

### C. Definition of 0/1 Knapsack Problem

The knapsack problem first appeared in the 1950s by George Dantzig (1957), the developer of linear programming. He showed that the knapsack problem is immensely studied and no doubt a commonly known optimisation problem. There are several types of knapsack problems, such as the 0/1 knapsack problem, unbounded knapsack problem, and fractional knapsack problem [18] [19]. In this paper, we would like to discuss one commonly known knapsack problem, the 0/1 knapsack problem. It has a value of either 0 or 1, and its dataset is a collection of items, where each type of item has its weight and value that is later used to determine the priority. The total cost is less or equal, and the real profit is as maximum as possible. The items are indivisible in the 0/1 knapsack problem; take all the value or leave it [2]. The 0/1 knapsack problem can be viewed as a sequence of decisions [1]. It was known as an NP-hard problem where the problem can be solved by fulfilling its constraint, which means taking it as a whole or none [10]. Other than the classic 0/1 knapsack problem, there also have been several types of the 0/1 knapsack problem, such as the discounted 0/1 knapsack problem [15] [16].

It had been studied of the knapsack problem, and many algorithms have been suggested to obtain a solution for the optimisation problems, such as dynamic programming, the whale optimisation algorithm, a binary monarch butterfly optimisation algorithm, the monkey algorithm, neural selection, etc. [12] [13] [14] [20]. The 0/1 knapsack problem also can be resolved through several techniques, such as divide-and-conquer recursively, incremental, etc. [17]. In this paper, we would like to discuss the general dynamic programming approach to the 0/1 knapsack problem.

There are also many implementations for 0/1 knapsack problems like cryptography, networking, production cost control, searching journals, decision-making, and resource allocation [2] [11].

### D. Dynamic Programming Algorithm Concept for 0/1 Knapsack Problem

Dynamic programming can be used to solve optimisation problems. For this matter, it was solving the 0/1 knapsack problem. In this case, dynamic programming is used to solve the 0/1 knapsack problem as an exact method solution [11] by considering the weights and values of each n item with a weight limit of W. The point is to get the maximum total value, which we can get by considering two arrays of integers of values and weights of each n item and integer of the weight limit (W). Using a dynamic programming algorithm, find out the optimal or maximum value of a set of v[] that the total w[] is smaller or equal to W. Since it is a 0/1 Knapsack problem, it's only

possible to take the entire item or leave it altogether. It can't be fractional.

TABLE I ITEM LIST

Item [i]	1	2	3	4
Value	100	20	60	40
Weight	3	2	4	1

Total items = 4

Total weight limit W = 5

To implement the algorithm, create a table where i denotes the number of items and w means the item's weight. Because the weight limit is 5, the column of the weight row has a total weight limit of W, as shown in Table II.

TABLE II CONTENT TABLE

V[i, w]	0 [w]	1	2	3	4	5
0 [i]						
1						
2						
3						
4						

The solution of the 0/1 knapsack problem with dynamic programming can be solved through:

IF weightMax[j] > weight THEN  
 $O[j, \text{weight}] = O[j-1, \text{weight}]$

ELSE IF weightMax[j] <= weight THEN  
 $O[j, \text{weight}] = \max(O[j-1, \text{weight}], v[j] + v[j-1, \text{weight} - \text{weightMax}[j]])$

Indications:

weightMax = weight limit / weight max

weight = weight of each item

O = optimal solution

v = value of each item

j = number of arrays

Table III shows the results of the dynamic programming algorithm for 0/1 knapsack problems from the solution above. The maximum value is 140, and the items taken are 1 and 4.

TABLE III SOLUTION

V[j, w]	O [w]	1	2	3	4	5
O [j]	0	0	0	0	0	0
1	0	0	0	100	100	100
2	0	0	20	100	100	120
3	0	0	20	100	100	120
4	0	40	40	100	140	140

### E. Dynamic Programming Algorithm Design for 0/1 Knapsack Problem

From the passage above, a dynamic programming algorithm can be implemented through:

```
knapsack(WeightMax,weight,value):
    n = len(value)
    V = [[0 for j in range(WeightMax)] for k in range(value)]

    for h in range(1, WeightMax):
        for m in range(1, value):
            V[m][n] = V[m-1][n]
            if weight[m] < w and V[m-1][n-weight[m]] + v[m]
            > V[m][n]:
                V[m][n] = V[m-1][n-weight[m]] + v[m]

    return V
```

Once the optimal combination of items at some weight  $w < \text{WeightMax}$  is solved and several things  $i < n$ , the problem can be obtained by checking the higher item's weight by restricting the optimal solution provided by the dynamic programming algorithm (sub problems' optimal solution). The best solution to the 0/1 knapsack problem can be obtained through the optimal solutions of each subproblem. It makes the process more efficient and thorough.

### F. Time Complexity of Dynamic Programming Algorithm for 0/1 Knapsack Problem

The time complexity of the dynamic programming algorithm for the 0/1 knapsack problem is  $O(n^2)$ . The calculation of  $n^2$  can be obtained by multiplying the range number of 0 by WeightMax and the range number of items. It created the matrix of  $V(i,w)$ .

## III. PROPOSED IDEAS

Knapsack problems can be found easily in our daily lives, especially 0/1 knapsack problems. It can be helpful in various areas. The industry is one area to consider for implementing dynamic programming in 0/1 knapsack problems. As already mentioned, determining prices counts as a 0/1 knapsack problem. In the furniture industry, to search for the appropriate price for the customer by calculating the production cost that involves material cost. Furniture industries must know every type of wood to be used as their materials in producing furniture, along with their prices. One of the woods that's usually used as home furniture is plywood. The table below is the list of types of plywood with their values and weight.

TABLE IV. ITEM LIST

No.	Types of Plywood	Wood's Values	Wood's Weight
1	Particle Board	100	4
2	Multiplex	115	8
3	Blockboard	155	12

No.	Types of Plywood	Wood's Values	Wood's Weight
4	Teakblock	190	16
5	Medium Density Fiberboard	205	20
6	Melamine	245	20

The table above shows many types of plywood with different values to indicate which one should be used. The more excellent value may lead to a better kind of wood, and the most significant discount from the table is Melamine, with the highest value of 245. Each type has its endurance, quality, and function. The best quality melamine has a softer texture and is often called decorative plywood. The second-best quality is a medium-density fiberboard that has a flatter surface, and its quality is also good. The third best quality is the teak block, a wood with excellent endurance and various patterns. Next, the blockboard is relatively thinner than the other, but it's substantial and multiplex, often called waterproof plywood, is very solid and can last for many years. Finally, Particle Board is the cheapest, vulnerable to less sturdy water.

Therefore, to determine which wood combination should be chosen as the material to produce home furniture like a kitchen set, a dynamic programming algorithm can be used in this 0/1 knapsack problem. For example, the weight limit must be 40 to build one kitchen. Given the maximum weight and each wood's value and importance, we could run the dynamic programming algorithm based on it. The weight will be counted by a multiple of 4 to be more effective.

TABLE V. SOLUTION

$V$ [j, w]	$\theta$ [w]	4	8	12	16	20	24	28	32	36	40
0 [0]	0	0	0	0	0	0	0	0	0	0	0
1	0	100	100	100	100	100	100	100	100	100	100
2	0	100	115	215	215	215	215	215	215	215	215
3	0	100	215	215	255	270	370	370	370	370	370
4	0	100	215	215	255	290	405	405	445	460	560
5	0	100	215	215	255	290	405	420	445	460	560
6	0	100	215	215	255	290	405	460	460	500	560

Table V shows that the optimal value from the data is 560. The plywood type that's included can be seen through:

```
IF V[p, w] = V[p-1, w] THEN
    flag = 0
    p = p-1

ELSE IF V[p, w] != V[p-1, w] THEN
    flag = 1
    p = p-1
    w = w-W[p]
```

The woods included are Particle Board, Multiplex, Blockboard, and Teakblock. The combination of those four types of plywood is the most optimal one rather than the other

possible combination. With a total weight of 40, the value reaches its finest in 560.

The cost of each piece of furniture can be determined after deciding which wood should be used to build the kitchen set. That is the simple way of using a dynamic programming algorithm in the industry's 0/1 knapsack problem.

In conclusion, the steps of the algorithm we are going to use are listed below:

1. Define the value and weight of each type of plywood.
2. Calculate how many items will be used as the base of the problem.
3. Create a table with the size of (total items + 1) x (maximum weight + 1).
4. Find the maximum value in each iteration to put in the table's cell.
5. The last cell will be the optimal value, and then from that cell, we will go backwards to determine which items will be included.

#### IV. IMPLEMENTATION

The approach to the problem stated before is considering two principal cases for every wood in the list—whether the wood is included or not in the optimal subset. Using dynamic programming, the same sub-issues can be computed only once by creating a bottom-up manner temporary array  $V[][]$ . Below is the implementation of the 0/1 knapsack problem in the furniture industry using C language.

```
#include <stdio.h>

int max(int item_x, int item_y)
{ if(item_x > item_y)
  { return item_x;
  }
  else
  {return item_y;
  }
}

void getKnapsack(int maxWeight, int wt[], int val[], int n, char wood[][30])
{
    int j, weight;
    int V[n+1][maxWeight+1];

    for(j=0; j<=n; j++)
    {
        for(weight=0; weight<=maxWeight; weight++)
        {
            if (j == 0 || weight == 0)
            {
                V[j][weight] = 0;
            }
            else if (wt[j-1] <= weight)
            {
                V[j][weight] = max(val[j-1] + V[j-1][weight - wt[j-1]], V[j-1][weight]);
            }
            else
            {
                V[j][weight] = V[j-1][weight];
            }
        }
    }
}
```

```
int result = V[j][maxWeight];
printf("Optimal Value: %d\n", result);
printf("Wood Combinations:\n");

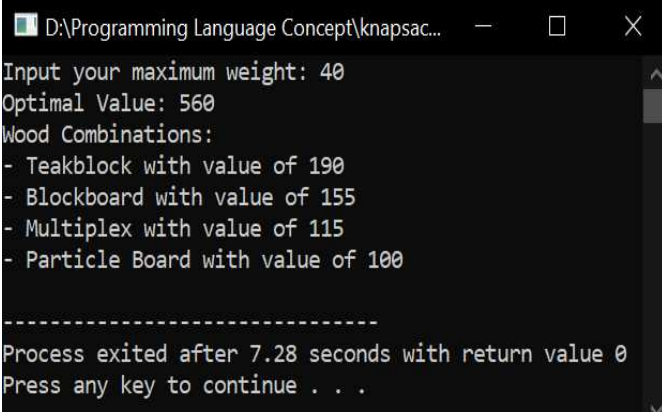
weight = maxWeight;
for(j=n; j>0 && result>0; j--)
{
    if(result == V[j-1][weight])
    {
        continue;
    }
    else
    {
        printf("- %s with value of %d\n", wood[j-1], val[j-1]);
        result = result - val[j-1];
        weight = weight - wt[j-1];
    }
}

int main()
{
    char wood[][30] = {"Particle Board", "Multiplex", "Blockboard", "Teakblock", "Medium Density Fiberboard", "Melaminto"};
    int val[] = {100, 115, 155, 190, 205, 245};
    int wt[] = {4, 8, 12, 16, 20, 20};
    int n = sizeof(val) / sizeof(val[0]);
    int maxWeight;

    printf("Input your maximum weight: ");
    scanf("%d", &maxWeight);

    getKnapsack(maxWeight, wt, val, n, wood);
    return 0;
}
```

First, we define our input value (val), weight (wt), and types of plywood (woods) all in their array. The user will input the maximum weight value before the program proceeds to the next step. Then, through the calling function getKnapsack, we implement the algorithm for the 0/1 knapsack problem. All of the variables we need are passed, and then the function starts to calculate and map the maximum value that doesn't exceed the maximum weight into the table/matrix ( $V[i][w]$ ).



```
D:\Programming Language Concept\knapsac...
Input your maximum weight: 40
Optimal Value: 560
Wood Combinations:
- Teakblock with value of 190
- Blockboard with value of 155
- Multiplex with value of 115
- Particle Board with value of 100

-----
Process exited after 7.28 seconds with return value 0
Press any key to continue . . .
```

Fig. 1. The output with a maximum weight of 40

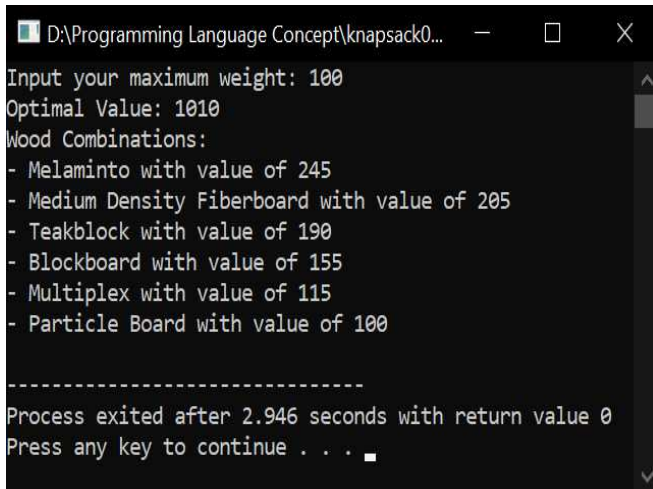
Function max will be called to get the maximum value, and the values we want to check will be compared. The function max will then return the most immense value. When the



mapping is done, the final result or the best deal will be revealed and put into a new variable (effect).

The variable result will determine which plywood types are included in the optimal subset. Using the same algorithm we've stated before, all sorts of included plywood will be shown later.

Finally, after running the code, the output would be like the one in Fig. 1. As shown in Fig. 1, the result is the same as Table V. Users can try with their designated maximum weight. For example, the user will enter 100 as their total weight; the output would look like the one in Fig. 2.



```

D:\Programming Language Concept\knapsack0...
Input your maximum weight: 100
Optimal Value: 1010
Wood Combinations:
- Melaminto with value of 245
- Medium Density Fiberboard with value of 205
- Teakblock with value of 190
- Blockboard with value of 155
- Multiplex with value of 115
- Particle Board with value of 100

-----
Process exited after 2.946 seconds with return value 0
Press any key to continue . . .

```

Fig. 2. The output with a maximum weight of 100

Overall, the program runs smoothly, and we can see the optimal value based on the weight limit given, along with the combinations of included plywood.

## V. CONCLUSION

Implementation of the dynamic programming algorithm through the 0/1 knapsack problem can be implemented to solve various optimisation problems. The furniture industry production cost is one of them. At the same time, the production cost is one of the most critical aspects of the business to run. The dynamic programming algorithm using the 0/1 knapsack problem helps to solve the problem of determining the best wood that can be used to achieve the maximum profit by considering the production cost and the value of the selling price.

Thus, the implementation of the dynamic programming algorithm using the 0/1 knapsack problem can be implemented not only for the production cost of the furniture industry but also for the production cost of many different areas of industry that require one whole of a material to be produced to get the result of products.

## ACKNOWLEDGEMENTS

This research was funded by the Directorate of Research, Technology, and Community Service, Directorate General of Higher Education, Research and Technology. Ministry of Education, Culture, Research and Technology, Following the

2023 Fiscal Year Research Contract No  
1402/LL3/AL.04/2023, Dated 26 June 2023

## REFERENCES

- [1] Chebil, K., & Khemakhem, M. (2015). *A dynamic programming algorithm for the Knapsack Problem with Setup*. *Computers & Operations Research*, 64, 40–50.
- [2] Shaheen, A., & Sleit, A. (2016). *Comparing between different approaches to solve the 0/1 Knapsack problem*. *International Journal of Computer Science and Network Security (IJCSNS)*, 16(7), 1–11.
- [3] Goyal, S., & Parashar, A. (2016). *A proposed solution to knapsack problem using branch & bound technique*. *International J. for Innovation Research Multidisciplinary Field*, 2(7), 240–246.
- [4] Haskell, W. B., Jain, R., & Kalathil, D. (2016). *Empirical Dynamic Programming*. *Mathematics of Operations Research*, 41(2), 402–429.
- [5] Saphiro, A., & Pichler, A. (2016). *Time and Dynamic Consistency of Risk Averse Stochastic Programs*.
- [6] Jiang, Y., & Jiang, Z.-P. (2015). *Global Adaptive Dynamic Programming for Continuous-Time Nonlinear Systems*. *IEEE Transactions on Automatic Control*, 60(11), 2917–2929.
- [7] Wang, Y., O'Donoghue, B., & Boyd, S. (2014). *Approximate dynamic programming via iterated Bellman inequalities*. *International Journal of Robust and Nonlinear Control*, 25(10), 1472–1496.
- [8] Wang, F. Y., Zhang, J., Wei, Q., Zheng, X., & Li, L. (2017). *PDP: Parallel dynamic programming*. *IEEE/CAA Journal of Automatica Sinica*, 4(1), 1–5.
- [9] Gao, W., & Jiang, Z. P. (2016). *Adaptive dynamic programming and adaptive optimal output regulation of linear systems*. *IEEE Transactions on Automatic Control*, 61(12), 4164–4169.
- [10] Ezugwu, A. E., Pillay, V., Hirasen, D., Sivanarain, K., & Govender, M. (2019). *A Comparative study of meta-heuristic optimisation algorithms for 0–1 knapsack problem: Some initial results*. *IEEE Access*, 7, 43979–44001.
- [11] Lv, J., Wang, X., Huang, M., Cheng, H., & Li, F. (2016). *Solving 0-1 knapsack problem by greedy degree and expectation efficiency*. *Applied Soft Computing*, 41, 94–103.
- [12] Abdel-Basset, M., El-Shahat, D., & Sangaiah, A. K. (2019). *A modified nature inspired meta-heuristic whale optimisation algorithm for solving 0–1 knapsack problem*. *International Journal of Machine Learning and Cybernetics*, 10(3), 495–514.
- [13] Feng, Y., Wang, G. G., Deb, S., Lu, M., & Zhao, X. J. (2017). *Solving 0–1 knapsack problem by a novel binary monarch butterfly optimisation*. *Neural computing and applications*, 28(7), 1619–1634.
- [14] Zhou, Y., Chen, X., & Zhou, G. (2016). *An improved monkey algorithm for a 0-1 knapsack problem*. *Applied Soft Computing*, 38, 817–830.
- [15] He, Y. C., Wang, X. Z., He, Y. L., Zhao, S. L., & Li, W. B. (2016). *Exact and approximate algorithms for discounted {0-1} knapsack problem*. *Information Sciences*, 369, 634–647.
- [16] Zhu, H., He, Y., Wang, X., & Tsang, E. C. (2017). *Discrete differential evolutions for the discounted {0-1} knapsack problem*. *International Journal of Bio-Inspired Computation*, 10(4), 219–238.
- [17] Chan, T. M. (2018). *Approximation schemes for 0-1 knapsack*. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
- [18] He, X., Hartman, J. C., & Pardalos, P. M. (2016). *Dynamic-programming-based inequalities for the unbounded integer Knapsack problem*. *Informatica*, 27(2), 433–450.
- [19] Ferdosian, N., Othman, M., Lun, K. Y., & Ali, B. M. (2016). *Optimal solution to the fractional knapsack problem for LTE overload-state scheduling*. In *2016 IEEE 3rd International Symposium on Telecommunication Technologies (ISTT)* (pp. 97–102). IEEE.
- [20] Shyamala, K., & Chanthini, P. (2017). *A novel approach in solving 0/1 Knapsack Problem using neural selection principle*. In *2017 IEEE International Conference on Power, Control, Signals, and Instrumentation Engineering (ICPCSI)* (pp. 2601–2605). IE