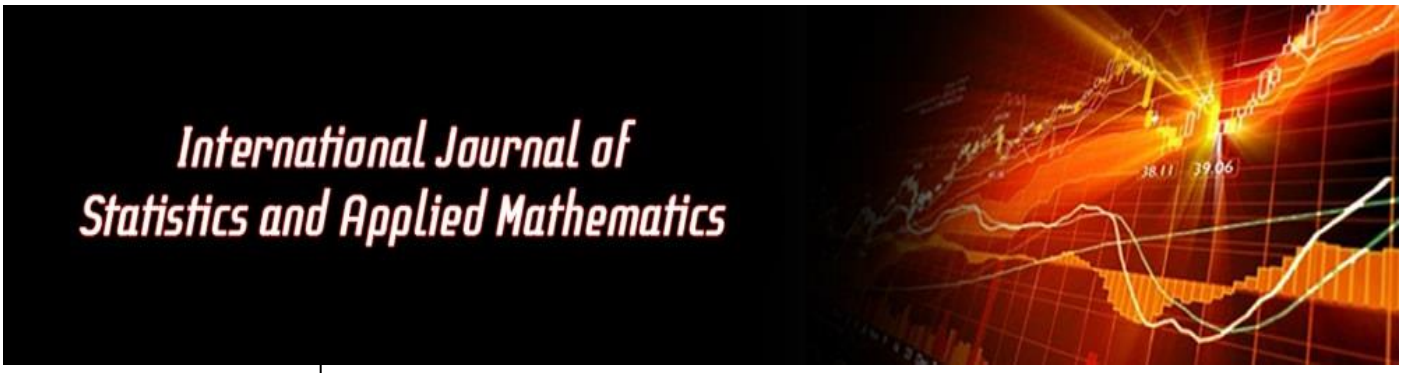
International Journal of Statistics and Applied Mathematics 2019; 4(3): 54-62



ISSN: 2456-1452
Maths 2019; 4(3): 54-62

www.mathsjournal.com
Received: 25-03-2019
Accepted: 27-04-2019

Burcu Durmuş
Department of Statistics, Science of Faculty, Muğla Sıtkı Koçman University, Muğla, Turkey

Öznur İşçi Güneri
Department of Statistics, Science of Faculty, Muğla Sıtkı Koçman University, Muğla, Turkey

Nevin Güler Dincer
Department of Statistics, Science of Faculty, Muğla Sıtkı Koçman University, Muğla, Turkey

Correspondence
Öznur İşçi Güneri
Department of Statistics, Science of Faculty, Muğla Sıtkı Koçman University, Muğla, Turkey

Greedy solution method for knapsack problems with R

Burcu Durmuş, Öznur İşçi Güneri and Nevin Güler Dincer

Abstract
The Knapsack problems contained in integer programming are one of the combinatoral optimization methods that maximize utility without exceeding the capacity. Different methods have been developed for the solution of Knapsack problems. In this study, Branch boundary algorithm, Balas algorithm and Greedy algorithm are used. Among these, R program code was developed for Greedy algorithm and sample problems were solved in different models. Results were also obtained using the pomQM package program for 3, 5, 7, 10, 30 and 50 variables. All results were compared using the developed R code. The objective function values, constraint values and capacity values of the problems are produced by random in R program. Even when the number of variables for single and two-dimensional, pure (0-1) binary problems is large, it can be obtained optimal or close to optimality results with Greedy method. As a result, R-code was developed without any problem of size and number of constraints.

Keywords: 0-1 knapsack problem, pure knapsack problem, greedy algorithm, R program

1. Introduction
Optimization problems cannot be used for many reasons in today's enterprises. The main reason for this is that the problems cannot be expressed algebraically or the number of constraints is high. In such cases, the choice of heuristic algorithms will help to solve the problem with less cost and therefore will be of great benefit for the enterprise. Artificial neural networks, genetic algorithms, tabu search, ant colony and Greedy algorithm are widely used heuristic algorithms. Apart from these, there are many intuitive or meta intuitive algorithms developed to solve the problem. However, the Greedy algorithm has the advantage that it has less computational cost compared to other heuristic algorithms, can be solved by using less algorithms and can be applied to any problem. The use of the Greedy algorithm for local or instant results is a matter of debate, but its use is quite common.

The reason why Greedy algorithm is preferred for heuristic algorithms is that it has such advantages. In addition, the Greedy algorithm, which is included in many algorithms, helps to obtain healthier results with this algorithm and to develop new hybrid algorithms. For example, among the classical methods, the most commonly used branch boundary algorithm reduces the scope of the problem by trimming the branches and decreases the processing load. Or, it can be more economical to find the optimal point for finding the randomly selected point with the Greedy approach in the cutting plane method. The Greedy algorithm is not only useful in classical algorithms, but also in heuristic algorithms. Greedy has been found to be of great benefit in the determination of the population to be randomly assigned to the genetic algorithm solution, and the creation of a banned list in taboo search (Gerşil & Palamutçuoğlu, 2013; Alpay, 2008) [1, 2].

In this study, the solution of knapsack problems with Greedy algorithm is discussed. For this purpose, the sample problems were solved and solved with the help of the classical methods of the branch boundary algorithm and the Balas algorithm and the R program codes developed based on the Greedy algorithm (R Development Core Team, 2018) [3].

Different softwares are used in the literature to solve integer programming problems. However, these softwares are usually limited by the number of variables and iterations. For this reason, writing code for problem solving is important in order to eliminate both the variable and iterative constraints and to obtain the solutions in a shorter time with the desired methods.

In addition, typed codes allow to solve new problems encountered by making arrangments on them. Considering these situations, it is preferred to make the solution with R which is one of the most used programs in statistics.

2. Review of literatüre

The knapcask problems are one of the most common integer programming problems. Studies on integer programming are usually based on classical algorithms or heuristic algorithms that have come to the fore in recent years. The Greedy algorithm included in these algorithms is preferred for problems that other algorithms cannot solve or this intuition is used in some algorithms, which has increased the importance of the algorithm. Especially its yielding the best results for the problems that can be expressed as matroid has increased the confidence in Greedy's intuition.

When the studies on the solution of integer programming problems with the Greedy algorithm were examined, Lagodimos and Leopoulos (2000) [4] first pointed out the problem of planning the personnel shifts of a food company. They addressed the problem under certain constraints and solved the problem with the proposed Greedy algorithm. As a result, the algorithm performed well in terms of both solution time and quality.

In a study to align DNA sequences, Zhang *et al* . (2000) [5] concluded that the Greedy algorithm provided a tenfold resolution of the appropriate data compared to the traditional programming. They stated that an application of the algorithm in the study was used in the UniGene database.

Biller *et al* . (2005) [6] used the Greedy algorithm in their study in the automotive industry and showed that it gave the optimal solution. At the end of the study, they stated that with a few price changes, much better benefits could be obtained.

Berberler (2009) [7] conducted a doctoral study on the types of knapsack and their application. In the study, the problem of can knapsack was dealt with knapsack terminology and solved with the help of the genetic algorithm to obtain optimum results for the knapsack.

Liu (2001) [8] conducted a simulation study on the solution of 0-1 knapsack problems using Greedy and dynamic programming methods.

Gorski *et al* . (2012) [9] showed that two-dimensional knapsack problems can be solved efficiently by the Greedy algorithm. For this purpose, they have appropriately divided the weightings in the constraints. They also stated that the Greedy method produced suitable solutions in a shorter time.

When the studies done so far are examined, it is see that though different programs (WinQSB, pomQM, Gams, etc.) have been developed for the solution of knapsack problems, the number of commonly used package programs is low. In addition, packet programs developed for classical methods are limited by the number of constraints and number of variables. Apart from this, studies to solve such problems with an heuristic algorithm are insufficient. Therefore, it is important to write code with a commonly used package program like R now. With the developed R program codes, solutions can be found for 0-1 and pure knapsack problems in shorter time without variable number constraint.

3. Materials and methods

Let's assume that someone has n number of unbrokable products and that these products are intended to be filled into a knapsack with a limited volume. Each of these products has a specific c_{ij} benefit (utility) value for the knapsack owner and each of them has an existing a_j weight (j = 1, 2, …, n). Provided that the b_j capacities do not exceed, the knapsack owner would like to create a combination of products that will provide the most benefit to him. The decision variables x_j are the number of j products to be taken into the knapsack. In terms of the definition of these decision variables, knapsack problems are classified as non-negative knapsack and (0-1) binary knapsack problems. So a general knapsack model is expressed as follows (Winston, 2014) [10];

Objective function:

$$Max\ Z = \sum_{j=1}^{n} c_j x_j \tag{1}$$

Subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_j$$

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

$$x_j = 0\ and\ integer$$

The model given by the inequality under the above constraints (1) is called the non-negative or pure knapsack problem. If the x_j limit is changed to 0 or 1, the problem becomes a 0-1 knapsack problem. In this case the model is expressed as follows (Schrijver) [11];

Objective function

$$Max\ Z = \sum_{j=1}^{n} c_j x_j$$

Subject to: (2)

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_j$$

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

$$x_j = 0 \; or \; 1$$

In this case, the decision variables of the inequality in the knapsack problem (2) take the value of 0-1. In the model given with the inequality (1), it is possible to put more than one product of the same type into the bag. However, the decision variables in this model should be defined in the form of 0-1 to determine whether to put the *j*th product in the knapsack (Özkan) [12].
The problem of the knapsack where some of the variables take integers and others take real values turns into a mixed integer problem model. In this case the model;
Objective function:

$$Max\ Z = \sum_{j=1}^{n} c_j x_j \quad (3)$$

Subject to

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_j$$

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

$$x_j \geq 0 \; and \; integer \; j = 1,2,\dots,p$$

$$x_j \geq 0 \; j = p,\dots,n \; (p \leq n)$$

(3) numbered is like inequality. Numerous algorithms have been developed to solve integer programming problems. Some of the most common algorithms used in the solution of knapsack problems are given below.

3.1. Branch and boundary algorithm

One of the most popular methods for integer programming problems, the branch and boundary algorithm has been in use since the 1960s. This algorithm is among the classical solution methods. The method searches for optimal solutions to the problem and ultimately determines the optimum value. The problem is initially addressed as a linear programming problem. At this stage, the branching process is started by looking at the real valued results. The branching phase continues until the integer results are found. When searching for solutions, the branching is terminated when the appropriate solution cannot be found. The branch and boundary algorithm can be compared to the decision trees in terms of shape.
The branch and boundary algorithm is a suitable method for many problems, but it should be remembered that it has exponential complexity. To reduce this complexity, the branch boundary algorithm can be used in combination with Gomory trimming or an heuristic method (Greedy algorithm, genetic algorithm, etc.) some branches can be pruned.

3.2. Balas algorithm

Balas algorithm 0-1 is used for integer problems. All the possible results with the algorithm are searched for optimal solution. When applying the method, the values 0 and 1 are matched for all variables. If we want to express mathematically, all variables are determined to be 0 and 1. This means that there are $2^{\text{number of variables}}$ of solutions for the problem. The optimal result is the solution, but for large-scale problems this method is useless.

3.3. Greedy algorithm

The Greedy algorithm is an heuristic method. This algorithm consciously or unconsciously is the most widely used method in daily life. It aims to find local results, not an optimal result. According to the objective function, it will try to find the values that will make the benefit maximum or the cost minimum by using very few and sometimes no algorithms. Calculation costs are quite low. The biggest advantage in comparison with other heuristics is that it finds a solution for every problem. Nevertheless, it should be noted that these solutions may not be optimal (Durmuş, 2018; Pan & Zhang, 2018) [13, 14].
The use of the Greedy algorithm in other methods both reduces the costs of the methods and allows the emergence of new hybrid algorithms. Considering the advantages of the Greedy algorithm, its importance is better understood. Therefore, R codes written in the study were written with Greedy terminology.

4. Results

In this section, classical algorithms for different knapsack problems and pomQM package program have been obtained. The same problems were solved by using R program codes developed using the Greedy algorithm. The main goal here is to show that local results can be obtained by the Greedy algorithm.

In literature, knapsack problems are usually single-conditional to "weight" or "volume"; Sometimes a multi-conditional mathematical model is created in terms of "weight" and "volume". For this reason, both single and two restricted knapsack problems were discussed in our study. Since three or more restricted knapsack problems were not found in the literature, they were not included in the study.

4.1. One dimensional 0-1 knapsack problem

An example of 8 variables is given below to one dimensional 0-1 knapsack problem.

Objective function

$$Max\ Z = x_1 + 5x_2 + 10x_3 + 3x_4 + 8_{x5} + 12x_6 + 9x_7 + 16x_8$$

Subject to (4)

$$20x_1 + 50x_2 + 100x_3 + 150x_4 + 70x_5 + 200x_6 + 150x_7 + 30x_8 \leq 500 x_j = 0\ or\ 1$$

When the problem given by the above (4) inequality is solved by the branch method from the classical methods, the results of each branch must be calculated separately. Results for this example were found to be $x_1 = x_1 = x_3 = x_5 = x_6 = x_8 = 1, x_4 = x_7 = 0$ and $Z = 52$.

When the problem is solved by Balas method, 2^8 solutions are obtained. These solutions also include optimal results. As a result, the optimal value is the same as the branch and boundary algorithm.

The following algorithm is followed to solve the problem with Greedy (Pan & Zhang, 2018) [14]:

Step 1. All the benefit values $r_j = (c_j/a_j)$ are calculated (knapsack is the maximization problem)

Step 2. The utility values (r_j) are sorted. If there is the same benefit value, their order is randomly determined.

Step 3. The most benefit item is put in the bag. The next benefit value is considered. If it does not exceed the capacity, it is placed in the bag and if it exceeds, the next item is checked.

Step 4. If all items have been checked, the solution is terminated. Otherwise, return to step 3.

R program codes are given in Table 1 for Greedy algorithm.

Table 1: R program codes for one dimensional 0-1 knapsack problem

```
one_knapsack_01 <- function(object, subject, capacity)
{ variable <- c (1:length(object))
benefit <- object/subject
place <- order(benefit, decreasing = T)
matrix <- cbind(variable, object, subject, benefit, place)
benefit_matrix <- matrix[order(matrix[,4], decreasing = T),]
x <- 1
final = 0
s <- 0
array <- c(x)
while (nrow(matrix) >= x)
{{if(benefit_matrix[x,3] <= subject)
{final = final + benefit_matrix[x,2]
capacity = capacity-benefit_matrix[x,3]
array[x] = s+1}
x <- x+1}}
array_plus <- cbind(benefit_matrix, array)
array_order <- array_plus[order(array_plus[,1], decreasing = F),]
print(paste("Z=", final))
print("xi values:")
print(array_order[,6]) }
```

According to these steps, the Greedy algorithm solution of the knapsack problem is given in Table 2.

Table 2: Benefits of a one dimensional 0-1 knapsack problem

x_j	c_j	a_j	$r_j=c_j/a_j$	rank
x_1	1	20	0.05	7
x_2	5	50	0.1	3
x_3	10	100	0.1	4
x_4	3	150	0.02	8
x_5	8	70	0.11	2
x_6	12	200	0.06	5
x_7	9	150	0.06	6
x_8	16	30	0.53	1

Solution is found as $x_1 = x_1 = x_3 = x_5 = x_6 = x_8 = 1, x_4 = x_7 = 0 \; and \; Z = 52$. The solution of the problem of 0-1 knapsack problem given by the equality (4) with the R program is as in Table 3;

Table 3: R program solution of one dimensional 0-1 knapsack problem

```
> object <- c (1, 5, 10, 3, 8, 12,9,16)> subject <- c (20, 50, 100, 150, 70, 200, 150, 30) > capacity <- 500
> one_knapsack_01(object, subject, capacity)
[1] "Z= 52"
[1] "xj values:"
[1] 1 1 1 0 1 1 0 1
```

4.2. One dimensional pure knapsack problem

An example of 10 variables is given below to one dimensional pure knapsack problem.
Objective function:

$$Max\ Z = 6x_1 + x_2 + 10x_3 + 6x_4 + 3_{x5} + 12x_6 + 7x_7 + 2x_8 + x_9 + 12x_{10} \quad (5)$$

Subject to:

$$2x_1 + x_2 + 2x_3 + 3x_4 + x_5 + x_6 + 5x_7 + 2x_8 + 3x_9 + 4x_{10} \leq 24$$
$$x_j \geq 0 \; and \; integer$$

When the problem given by the above (5) inequality was solved by the branch boundary algorithm, the results were found as $x_6 = 24, x_1 = x_2 = x_3 = x_4 = x_5 = x_7 = x_8 = x_9 = x_{10} = 0$ and $Z = 288$. Because the problem is pure integer programming, the solution with the Balas algorithm cannot be applied to this problem.
The following algorithm steps can be defined to solve the problem with Greedy:

Step 1. All the benefit $r_j = (c_j/a_j)$ values are calculated.

Step 2. The utility values are sorted. If there is the same benefit value, their order is randomly determined.

Step 3. The item that yields the most benefit is put into a bag not to exceed the capacity The next benefit value is considered. If it does not exceed the capacity, it is put in a bag not to exceed the capacity and if it exceeds, the next item is checked.

Step 4. If all items have been checked, the solution is terminated. Otherwise, return to step 3.
R program codes are given in Table 4.

Table 4: R program codes for one dimensional pure knapsack problem

```
one_knapsack_pure <- function(object, subject, capacity)
{ variable <- c (1:length(object))
benefit <- object/subject
place <- order(benefit, decreasing = T)
matrix <- cbind(variable, object, subject, benefit, place)
benefit_matrix <- matrix[order(matrix[,4], decreasing = T),]
x <- 1
final = 0
s <- 0
array <- c(x)
while (nrow(matrix) >= x)
{{if(benefit_matrix[x,3] <= capacity)
{while(benefit_matrix[x,2]<= capacity)
{final = final + benefit_matrix[x,2]
capacity = capacity-benefit_matrix[x,3]
array[x] = s+1
s <- s+1}}
x <- x+1
s <- 0}}
```

```
array_plus <- cbind(benefit_matrix, array)
array_order <- array_plus[order(array_plus[,1], decreasing = F),]
print(paste("Z=", final))
print("xi values:")
print(array_order[,6]) }
```

According to these steps, the solution of the knapsack problem is as in Table 5.

Table 5: Benefit values of one dimensional pure knapsack problem

x_j	c_j	a_j	$r_j=c_j/a_j$	rank
x_1	6	2	3	3
x_2	1	1	1	8
x_3	10	2	5	2
x_4	6	3	2	6
x_5	3	1	3	4
x_6	12	1	12	1
x_7	7	5	1.4	7
x_8	2	2	1	9
x_9	1	3	0.33	10
x_{10}	12	4	3	5

Thus, the Greedy solution for the problem is found as $x_6 = 24, x_1 = x_2 = x_3 = x_4 = x_5 = x_7 = x_8 = x_9 = x_{10} = 0$ and $Z = 288$. The solution of one dimensional pure knaksack problem by R program is given in Table 6.

Table 6: R program solution of one dimensional pure knapsack problem

```
> object <- c (6, 1, 10, 6, 3, 12, 7, 2, 1, 12)
> subject <- c (2, 1, 2, 3, 1, 1, 5, 2, 3, 4)
> capacity <- 24
> one_knapsack_pure(object, subject, capacity)
[1] "Z= 288"
[1] "xj values:"
[1] 0 0 0 0 0 24 0 0 0 0
```

4.3. Two-dimensional 0-1 knapsack problem

The following problem is solved based on the differences between the benefit values and the effect on the capacity and the results are given (Bakır & Altunkaynak, 2003) [15].
Objective function:

$$Max\ Z = 35x_1 + 85x_2 + 135x_3 + 10x_4 + 25_{x5} + 2x_6 + 94x_7 \quad (6)$$

Subject to:

$$2x_1 + 3x_2 + 9x_3 + 0.5x_4 + 2x_5 + 0.1x_6 + 4x_7 \leq 25$$
$$15x_1 + 35x_2 + 105x_3 + 68x_4 + 125x_5 + 25x_6 + 100x_7 \leq 400$$
$$x_j \geq 0\ or\ 1$$

The problem given by (6) inequality is 2 dimensional and 7 variables. The benefit values for solving this problem with the Greedy algorithm are given in Table 7.

Table 7: Benefit values of the two-dimensional 0-1 knapsack problem

x_j	c_j	a_{1j}	a_{2j}	r_{1j}	r_{2j}	r_{1j}- r_{2j}	rank
x_1	35	2	15	17.5	2.33	15.17	5
x_2	85	3	35	28.3	2.42	25.88	1
x_3	135	9	105	15	1.28	13.72	6
x_4	10	0.5	68	20	0.14	19.86	4
x_5	25	2	125	12.5	0.20	12.30	7
x_6	2	0.1	25	20	0.08	19.92	3
x_7	94	4	100	23.5	0.94	22.56	2

The solution of the decision variables given in Table 5 according to the steps of Greedy algorithm is given in Table 8.

Tablo 8: Greedy solution of one dimensional 0-1 knapsack problem

x_j	Weight (≤ 25)	Volume (≤ 400)
x_2=1	25-3=22	400-35=365
x_7=1	22-4=18	365-100=265
x_6=1	18-0.1=17.9	265-25=240
x_4=1	17.9-0.5=17.4	240-68=172
x_1=1	17.4-2=15.4	172-15=157
x_3=1	15.4-9=6.4	157-105=52
x_5=0		

The solution of the problem with the Greedy algorithm was found to be $x_5 = 0, x_1 = x_2 = x_3 = x_4 = x_6 = x_7 = 1$ and $Z = 361$. The solution of the problem with the branch and boundary algorithm, Balas algorithm and R program were found to be the same. R program codes are given in Table 9 and the result of the R program is given in Table 10.

Table 9: R program codes for two-dimensional 0-1 knapsack problem

```
two_knapsack_01 <- function(object, subject1, subject2, capacity1, capacity2)
{ variable <- c(1:length(object))
benefit1 <- object/subject1
benefit2 <- object/subject2
benefit <- benefit1 - benefit2
place <- order(benefit, decreasing = T)
matrix <- cbind(variable, object, subject1, subject2, benefit, place)
benefit_matrix <- matrix[order(matrix[,5], decreasing = T),]
x <- 1
final = 0
s <- 0
array <- c(x)
while (nrow(matrix) >= x)
{ if(benefit_matrix[x,3] <= capacity1 && benefit_matrix[x,4] <= capacity2)
{final = final + benefit_matrix[x,2]
capacity1 = capacity1-benefit_matrix[x,3]
capacity2 = capacity2-benefit_matrix[x,4]
array[x] = s+1}
x <- x+1}
array_plus <- cbind(benefit_matrix, array)
array_order <- array_plus[order(array_plus[,1], decreasing = F),]
print(paste("Z=", final))
print("xi degerleri:")
print(array_order[,7]) }
```

Table 10: R program solution of two-dimensional 0-1 knapsack problem

```
> object <- c (35, 85, 135, 10, 25, 2, 94)
> subject 1<- c (2, 3, 9, 0.5, 2, 0.1, 4)
> subject 2<- c (15, 35, 105, 68, 125, 25, 100)
> capacity 1 <- 25
> capacity 2 <- 400
> two_knapsack_01(object, subject 1, subject 2, capacity 1, capacity 2)
[1] "Z= 361"
[1] "xj values:"
[1] 1 1 1 1 0 1 1
```

4.4. Two-dimensional pure knapsack problem

The problem given by equation (6) above can be re-modeled as pure knapsack. The Greedy solution for the newly defined problem was found to be $x_2 = 8, x_6 = 4, x_1 = x_3 = x_4 = x_5 = x_7 = 0$ and $Z = 688$. R program codes are given in Table 11. The solution of the problem by R program is given in Table 12. The same results were obtained in the solution of the problem by the branch and boundary algorithm. Since the problem is pure integer programming, the Balas solution cannot be applied.

Table 11: R program codes for two-dimensional pure knapsack problem

```
two_knapsack_01 <- function(object, subject1, subject2, capacity1, capacity2)
{ variable <- c(1:length(object))
benefit1 <- object/subject1
benefit2 <- object/subject2
benefit <- benefit1 - benefit2
place <- order(benefit, decreasing = T)
matrix <- cbind(variable, object, subject1, subject2, benefit, place)
benefit_matrix <- matrix[order(matrix[,5], decreasing = T),]
x <- 1
final = 0
```

```
s <- 0
array <- c(x)
while (nrow(matrix) >= x)
{ if(benefit_matrix[x,3] <= capacity1 && benefit_matrix[x,4] <= capacity2)
{while(benefit_matrix[x,3] <= capacity1 && benefit_matrix[x,4] <= capacity2)
{final = final + benefit_matrix[x,2]
capacity1 = capacity1-benefit_matrix[x,3]
capacity2 = capacity2-benefit_matrix[x,4]
array[x] = s+1
s <- s+1}}
x <- x+1
s <- 0}}
array_plus <- cbind(benefit_matrix, array)
array_order <- array_plus[order(array_plus[,1], decreasing = F),]
print(paste("Z=", final))
print("xi degerleri:")
print(array_order[,7]) }
```

Table 12: R program solution of two dimensional pure knapsack problem

```
> object <- c (35, 85, 135, 10, 25, 2, 94)
> subject 1 <- c (2, 3, 9, 0.5, 2, 0.1, 4)
> subject 2 <- c (15, 35, 105, 68, 125, 25, 100)
> capacity 1 <- 25
> capacity 2 <- 400
> two_knapsack_pure(object, subject 1, subject 2, capacity 1, capacity 2)
[1] "Z= 688"
[1] "xj values:"
[1] 0 8 0 0 0 4 0
```

5. Conclusion

Ready-to-use software programs developed to solve integer programming problems are limited by the number of variables, constraints and iterations. Therefore, software development for the solution of problems is necessary for the solution of larger problems. Because in real-life problems the number of variables and constraints is high.

As mentioned earlier, this study was designed to investigate whether Greedy's heuristic algorithm gives close solutions compared to classical algorithms. In this respect, different models are discussed. The solutions of the problems were determined by the appropriate classical algorithm (pomQM package) and the Greedy algorithm (R program). Single and two dimensional knapsack problems can be solved with the help of developed R program. These program codes are not limited to constraint (max 30), variable (max 30) or iteration number (max 10.000) as in pomQM and similar package programs, so they can produce solutions for large problems.

Table 13 shows the results of pomQM and R program for 3, 5, 7, 10, 30 and 50 variables for single and two constrained problems. The objective function values, constraint values and capacity values of the problems discussed within the scope of the study were randomly generated by giving the minimum and maximum range in R. The variable numbers in the table have been chosen to be solved with pomQM. The 50-variable sample was added to the table to show that the pomQM package could not provide a solution.

Table 13 clearly shows that the Greedy algorithm provides optimal results or results very close to the optimal. In addition, when the number of constraints increases, it is seen that the results of the Greedy algorithm are slightly different from the optimal. However, these differences are negligible except for very risky problems (patient treatment, amount of medication, crisis management, etc.). In addition, it was observed that the pomQM package caused a number of iterations errors in some 30 variable models.

Table 13: PomQM and R Program Results

Knapsack Problem	Number of Constraints	Number of Variables	Classical Algorithm PomQM Program	Greedy Algorithm R Program
0-1 Integer	1	3	15	15
		5	19	19
		10	48	48
		20	29	29
		30	48	46
		50	-	66
	2	3	11	11
		5	6	6
		10	33	33
		20	221	178
		30	32	29
		50	-	15
Pure Integer	1	3	20	20
		5	34	34
		10	315	315

		20	30	30
		30	70	70
		50	-	320
	2	3	12	12
		5	6	6
		10	88	56
		20	470	356
		30	99	99
		50	-	11

As a result, when different models are considered, the results are good and the Greedy algorithm works well. The importance of this interpretation is better understood by the use of the Greedy algorithm in hybrid algorithms. However, it should be kept in mind that the Greedy algorithm may give optimal results in some models. Other codes written for the heuristic algorithm solution produce solutions for a single problem. For this purpose, the codes developed with Greedy terminology for knapsack problems are separated due to the fact that they do not contain any restrictions compared to other software codes, they can solve almost every problem and they are easy to use.

These codes can also be used for problems that are the same as the knapsack problem model such as capital budgeting problems, capacity problems, packingproblems, cargo loading problems. Thus, the study covers similar problems that are frequently encountered in the literature. For these problem models, the desired results can be obtained quickly and practically, especially for small problems. In the following studies, it is aimed to develop R solution for other heuristic algorithms and to make it a package program which makes heuristic solutions of algorithms. Thus, the results of the problem can be compared easily in terms of heuristic algorithms and many convenience will be provided for researchers.

6. References

1. Gerşil M. ve Palamutçuoğlu, T. Ders Çizelgeleme Probleminin Melez Genetik Algoritmalar İle Performans Analizi, Niğde Üniversitesi İİBF Dergisi. 2013; 6(1):242-262.
2. Alpay Ş. Tam Zamanlı Montaj Hatlarında Çok Amaçlı Karışık Model Sıralama İçin Rota Birleştirmeli Açgözlü Rassallaştırılmış Uyarlamalı Arama Yordamı, Eskişehir Osmangazi Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi. 2008; 21(1):153-170.
3. R Development Core Team. R: A Language and Environment for Statistical Computing.
4. Lagodimos, A.G. ve Leopoulos V. Greedy Heuristic Algorithms for Manpower Shift Planning, International Journal of Production Economics. 2000' 68:95-106.
5. Zhang Z, Schwartz S, Wagner L, ve Miller W. A Greedy Algorithm for Aligning DNA Sequences, Journal of Computational Biology. 2000; 7(1/2):203-214.
6. Biller S, Chan LMA, Simchi-Levi, D ve Swann, J. Dynamic Pricing and The Direct-To-Customer Model in The Automotive Industry", Electronic Commerce Research. 2005; 5(2):309-334.
7. Berberler ME. Sırt Çantası Problem Türleri ve Uygulamaları, Doktora Tezi, Ege Üniversitesi, Fen Bilimleri Enstitüsü, İzmir.
8. Liu L. Solving 0-1 Knapsack Problems by Greedy Method and Dynamic Programming Method", Advanced Materials Research. 2011; 282:570-573.
9. Gorski J, Paquete L, ve Pedrosa F. Greedy Algorithms for A Class of Knapsack Problems With Binary Weights, Computers & Operations Research. 2012; 39: 498-511.
10. Winston WL. Operations Research Applications and Algorithms, Canada: Brooks/Cole Cengage Learning, 2014.
11. Schrijver A. Theory of Linear and Integer Programming. Amsterdam: A Wiley-Interscience Publication.
12. Özkan Ş. Yöneylem Araştırması Nicel Karar Teknikleri, İstanbul: Nobel Akademik Yayıncılık, 1999.
13. Durmuş B. Tamsayılı Programlamada Klasik ve Greedy Sezgisel Algortima Sonuçlarının Karşılaştırılması, Yüksek Lisans Tezi, Muğla Sıtkı Koçman Üniversitesi, Fen Bilimleri Enstitüsü, Muğla, 2018.
14. Pan X, ve Zhang T. Comparison and Analysis of Algorithms Fort He 0/1 Knapsack Problem, Journal of Physics, 1069: 1-7.
15. Bakır MA. ve Altunkaynak B. Tamsayılı Programlama Teori Modeller ve Algoritmaları, Ankara: Nobel Yayın Dağıtım, 2003.