Application of Pigeon Inspired Optimization for Multi-dimensional Knapsack Problem

F. Setiawan*, A. Sadiyoko, and C. Setiardjo

ABSTRACT

The multi-dimensional knapsack problem (MKP) is a generalization of the classical knapsack problem, a problem for allocating a resource/subset of objects that maximize profit while not violating the capacity of the knapsack. The MKP has many practical applications in different areas and is classified as an NP-hard problem. An analytic method like a branch and bound and dynamic programming can yield the optimum solution, but its computational time growth is exponentially proportional to the problem's size. Some heuristics/metaheuristics procedure has been developed to obtain a near-optimal solution within reasonable computational times. In this paper, a pigeon-inspired optimization (PIO) is proposed for solving MKP. PIO is one of the metaheuristic algorithms that is classified in population-based swarm intelligence that is developed based on the behavior of the pigeon to find its home. However, it had gone far away from its home. In this paper, PIO implementation to solve MKP is applied to two different characteristic cases in a total of 10 cases. The result of implementing the two-best combination of parameter values for 10 cases compared to particle swarm optimization, intelligent water drop algorithm, and the genetic algorithm give satisfactory results.

Keywords: metaheuristics, multi-dimensional knapsack problem, pigeon inspired optimization, swarm intelligence.

Article Info: Received November 20, 2020; Revised March 17, 2021; Accepted June 7, 2021.

1. INTRODUCTION

The multi-dimensional knapsack problem (MKP) is a generalization of the knapsack problem, a problem for allocating a resource/subset of objects that maximize profit while not violating the capacity of knapsack (Martello and Toth, 1990). It explained the classical knapsack problem as suppose a hitch-hiker has to fill up his knapsack by selecting from among various possible objects that will give him maximum comfort. Another example, the knapsack problem will provide the optimal solution related to investment. Suppose that someone is going to invest a number of dollars and consider some possible alternative investments. Using this method, he can calculate profit gained from investment and how much money is required to invest.

Recently, knapsack problems are an interesting topic in research. Examples of industrial knapsack problems are capital budgeting, cargo loading, and cutting stock (Martello and Toth, 1990). Since MKP is a generalization of the classical knapsack problem, it is more practical, and it can model all of the problems that can be modeled by the knapsack problem. According to Haddar et al. (2016), MKP has been used as a model for many real applications such as cutting stock problems, project selection, cargo loading problems, capital budgeting, allocating processors and databases in a distributed computer system, and daily management of a satellite.

Since MKP is an NP-hard problem, numerous methods have been developed by the researchers that are classified into exact and approximation methods. Exact method, such as branch and bound algorithm (Shih, 1979; Vimont et al., 2008; Manzini and Speranza, 2012); dynamic programming (Gilmore and Gomory, 1966; Weingartner and Ness, 1967); hybrid dynamic programming methods (Bertsimas and Demir, 2002; Balev et al., 2008) could generate the optimal solution,

F. Setiawan* is with the Department of Industrial Engineering, Universitas Katolik Parahyangan, Indonesia (email: fransetiawan@unpar.ac.id).
* Corresponding author

but they can solve only instances of very limited size in an acceptable computation time. Heuristics/metaheuristics procedure could obtain a near-optimal solution within reasonable computational times compared to the analytic method. There are two classifications of approximation methods, heuristics, and metaheuristics. Heuristics and metaheuristics have recently become a focus of researchers for solving MKP (Haddar et al., 2016). The heuristic approach has been developed by Battiti and Vecchiolli (1994) and Freville and Plateau (1986). Heuristics are problem-dependent that are designed and applicable to a problem. The research on the application of metaheuristic method to solve MKP are tabu search (Vasquez and Hao, 2001; Dammeyer and Voss, 1993, Glover and Kochenberger, 1996; Vasquest and Vimont, 2005, Lai et al., 2018); genetic algorithm (Khuri et al., 1994; Chu and Beasley, 1998; Berberler et al., 2013; Martins et al., 2014); simulated annealing (Leung et al., 2012; Rezoug et al., 2015); ant colony optimization (Kong et al., 2008; Ke et al., 2010; Fingler et al., 2014); particle swarm optimization (Kong et al., 2006; Hembecker et al., 2007; Wan and Nolle, 2009; Chen et al., 2010; Ktari and Chabchoub, 2013); Tisna, 2013, Chih, 2015, Haddar et al., 2016); intelligent water drops (Shah-Hosseini, 2009), binary artificial algae algorithm (Zhang et al., 2016), binary multi-verse optimizer (Baseet et al., 2019), modified flower pollination (Basset et al., 2018).

This paper will propound a new evolutionary computation technique, Pigeon Inspired Optimization (PIO) (Duan and Qiao, 2014), to answer MKP. PIO is a novel bio-inspired computation algorithm, which was inspired by the homing characteristics of pigeons (Duan and Li, 2014). PIO has been used to solve continuous air robot path planning problems (Duan and Qiao, 2014), target detection task for Unmanned Aerial Vehicles (UAVs) at low attitude (Duan and Li, 2014), control parameter design for automatic carrier landing system (Duan and Deng, 2016), prediction control for unmanned air vehicles (Dou and Duan, 2016).

Studies about the application of PIO for solving MKP have been done by Bolaji et al., 2017 and Bolaji et al., 2020. Bolaji et al. (2017) proposed a binary PIO for solving MKP. They used n-bit binary string representation to represent the MKP solution and used the penalty function method to tackle the MKP's constraints. Bolaji et al. (2020) developed a modified binary PIO for solving MKP, which integrated a crossover procedure of evolutionary algorithm in order to improve the solution. The crossover procedure is embedded in the landmark operator in the PIO. The result was that the modified binary PIO was outperformed the binary PIO.

This research is the same as what is proposed by Bolaji et al. (2017). In this paper, we propose another way to represent the MKP solution and to handling the MKP's constraints. We do not use the special 'binary' term of the PIO; however, we use the random value and use the sigmoid function to convert the random value to 0-1 (binary). We use the drop/add item procedure and local heuristic method, namely Heuristic Undesirability (HUD), to handle the MKP's constraints instead of the penalty function. Another difference is that we use ten instances from Senyu and Toyada (1967) and Weingartner and Ness (1967) and compare the PIO performance from particle swarm optimization (PSO), intelligent water drops (IWD), and genetic algorithm (GA).

2. LITERATURE REVIEW

In this section, we describe multi-dimensional knapsack problem (MKP), pigeon-inspired optimization, and methods used in this research.

2.1. Multi-dimensional knapsack problem

The multi-dimensional knapsack problem (MKP) is a problem for allocating a resource/subset of objects to maximize profit without violating the capacity of knapsack (Martello and Toth, 1990). There are a set of n items which each item has m weight. Every item has a profit/a benefit p_i ($i = 1,....,n$), various weight W_{ji} for every single knapsack j, and there is a knapsack of m dimensions with a capacity of each $c_1,...., c_m$. The objective function is to choose a subset of items that assures the capacity of the knapsack is not exceed and yields maximum profit. The decision variable x_i represents whether the item i will be selected or not. It will value 1 if item i is selected and 0 if item i is not selected. MKP can be formulated as follow:

$$\text{Maximize } \sum_{i=1}^{n} p_i \times x_i \tag{1}$$

Subject to

$$\sum_{j=1}^{n} w_{ji} \times x_i \le c_j, \quad j = 1,...,m \tag{2}$$

$$x_i \in (0,1), \quad i = 1,...,n \tag{3}$$

2.2. Pigeon inspired optimization

Pigeon Inspired Optimization (PIO) is a population-based swarm intelligence metaheuristic that is inspired by pigeon's behavior. Inspired by these, in the PIO, the map and compass operator model is based on magnetic field and sun, whereas the landmark operator model is based upon landmarks (Duan and Qiao, 2014).

2.2.1. Map and compass operator

In map and compass operator, every t^{th} iteration will update positions, X_i, and velocities, V_i, of pigeon i in a D dimension search space (Duan and Qiao, 2014). The equations:

$$V_i(t) = V_i(t-1)e^{-Rt} + rand\left(X_g - X_i(t-1)\right) \tag{4}$$

$$X_i(t) = X_i(t-1) + V_i(t) \tag{5}$$

R = the map and compass factor.
$rand$ = random numbers from 0 to 1.
X_g = the most recent global best position compared with other positions among all pigeons.

2.2.2. Landmark operator

According to Duan and Qiao (2014), in landmark operator, 50% N_p diminishes the number of pigeons for every iteration. Nevertheless, the pigeons are not close enough to the destination and not familiar with the landmarks. $X_c(t)$ is a midpoint pigeon's position at t^{th} iteration. These equations show their update position for every iteration:

$$N_p(t) = \frac{N_p(t-1)}{2} \qquad (6)$$

$$X_c(t) = \frac{\sum X_i(t).fitness(X_i(t))}{N_p \sum fitness(X_i(t))} \qquad (7)$$

$$X_i(t) = X_i(t-1) + rand.(X_c(t) - X_i(t-1)) \qquad (8)$$

where N_p is the number of pigeons that is diminished at every generation, t is the t^{th} iteration, and X_c is the center of some pigeon's position at the t^{th} iteration, X_i is the position of i^{th} pigeon.

The fitness value in equation 7 can be altered based on the problem that will be solved. If the objective function is maximizing, then equation 9 will be used to calculate the fitness value; otherwise, equation 10 will be used. Equation 9 and 10 are given below:

$$fitness(X_i(t)) = \frac{1}{f\min((X_i(t)) + \epsilon^*)} \qquad (9)$$

$$fitness(X_i(t)) = f\max(X_i(t)) \qquad (10)$$

The details of PIO can be referred to by Duan and Qiao (2014).

3. METHODOLOGY

3.1. Encoding and decoding

Encoding and decoding is a process that is needed in applying PIO algorithm to solve MKP. Encoding is a process to translate the MKP problem to the PIO, and decoding is a process to translate the PIO inner modification process to the MKP problem solution. This PIO algorithm analogizes the position of pigeons in each dimension as a representation of the solution so that each pigeon is represented as a potential solution. The dimension of the pigeon in this study is items. The position of the pigeon will then be converted into binary numbers to determine whether that item is taken into the knapsack.

To find out more clearly how the encoding and decoding process in the application of PIO for MKP is explained through a simple example problem. Suppose company X will choose a project that will be done by company X in the next three years; each project has its own annual cost (the annual cost is analogous to the weight of each item). Each project also has its own profit that is obtained by the investment (the profit is analogous to the benefit of each item). Every year there is a maximum budget that is budgeted by company X to run the chosen project. The maximum budget is analogous to the capacity of knapsack (year 1 budget is capacity of knapsack 1, year 2 budget is capacity of knapsack 2, etc.). The maximum budget every year respectively is $25.000, $15.000 and $20.000. The budget and profit for each project per year can be seen in Table 1.

The encoding process is done by determining the position of the pigeon in each dimension in the 0th iteration. In this 0th iteration, the position will be obtained from a random value between 0-1 since there is no previous position (for the next iteration, it is adjusted with the position calculation formula in PIO). Table 2 shows the result of the random value that has been obtained for each dimension.

After the random value is obtained, then the decoding process is carried out. The decoding process is done by changing the random value into a binary number, where if it has a value of 1 then the decision of goods is taken and vice versa. Changing this binary number is done through the calculation of the sigmoid function. The sigmoid function is one method of converting continuous values into binary numbers in the application of metaheuristics (Banati and Bajaj, 2011 and Palit et al., 2011). The sigmoid function calculation formula is as follows:

$$X_{ij} = \begin{cases} 1, \text{if } rand() \le \dfrac{1}{1+\exp(-X_{ij})} \\ 0, \text{otherwise} \end{cases} \qquad (11)$$

Table 3 below shows the binary results using the sigmoid function equation for each pigeon position or the decision of which item is taken in each dimension.

3.2. Drop/add item

The decoding process then continues with the drop/add item procedure. This procedure is used to handle the constraints in MKP. The way to handle these

Table 1: Budget for each project per year

Project	Cost in year 1 (thousand $)	Cost in year 2 (thousand $)	Cost in year 3 (thousand $)	Profit (thousand $)
A	12	10	10	2000
B	8	5	5	1200
C	7	3	5	1000
D	10	7	10	1500

Table 2: Random value of the position of the pigeon in the 0th position

Dimension	D1	D2	D3	D4
Position	0,543	0,672	0,347	0,958

Table 3: Decoding using the sigmoid function

Dimension	D1	D2	D3	D4
Position	0,543	0,672	0,347	0,958
Binary	1	1	0	1

Table 4: HUD value of each project

Dimension	D1	D2	D3	D4
Position	0,543	0,672	0,347	0,958
Binary	1	1	0	1
HUD	0.0053	0.0050	0.0050	0.0060

Table 5: Problem characteristics for 10 cases

No	Instances	Number of Knapsack	Number of Item
1	SENTO1	30	60
2	SENTO2	30	60
3	WEING1	2	28
4	WEING2	2	28
5	WEING3	2	28
6	WEING4	2	28
7	WEING5	2	28
8	WEING6	2	28
9	WEING7	2	105
10	WEING8	2	105

constraints in this paper is by dropping (drop procedure) or adding (add procedure) the items. If there is excess capacity, the procedure that will be done is dropping the item. Otherwise, if there is remaining capacity, the procedure that will be done is adding the items as much as possible. The local heuristic method is used for handling this constraint. The local heuristic method used is Heuristic Undesirability (HUD). Based on Shah-Hosseini (2008), HUD is one of the simple local heuristic methods that show how big the item is undesirable in a solution. The formula for calculating the HUD for each item is as follow:

$$HUD(j) = \frac{1}{mb_j} \sum_{k=1}^{m} r_{jk} \qquad (12)$$

with b_j is the profit of item j, r_{jk} is the weight from item j at knapsack k, and m is the number of knapsacks. Based on equation 12 above, it is obtained the HUD value for each project/item. The procedure to handling the constraint in MKP can be done based on this value. The drop/add item procedure can be seen in the following flowchart in figure 1. Based on the flowchart, from the simple example about the choice of project in company X, the total weight of all items chosen exceeds the capacity/maximum budget. All three projects/items chosen (projects 1, 2 and 4) will be dropped one by one until the total weight of projects/items does not exceed the maximum budget/capacity from the item with the biggest HUD value. Table 4 shows the value of HUD for each project/item that is chosen for the drop item.

From table 4, it is known that the first project/item dropped is item 4 (has the biggest HUD value). After item 4 is dropped, it will be recalculated whether the project/item chosen exceeds the capacity/maximum budget. Capacity has been fulfilled when the project/item 4 is dropped, so the drop procedure process stops there. The procedure of drop/add item procedure can be seen in Figure 1.

3.3. Pigeon inspired optimization solving MKP

After generating the encoding and decoding process for applying pigeon inspired optimization to the multi-dimensional knapsack problem, the all step for applying the PIO in MKP that we proposed is explained in pseudocode shown in Figure 2.

4. RESULTS AND DISCUSSIONS

4.1. Numerical experiment instances

The MKP cases that will be solved using PIO are benchmark instances that are often used in other studies. The MKP instances have also been implemented using Particle Swarm Algorithm (PSO) (Hembecker et al., 2007), Intelligent Water Drops Algorithm (IWD) (Shah-Hosseini, 2009), and Genetic Algorithm (GA) (Khuri et al., 1994), and the result of these three of researches will be compared with the result from this research using PIO.

There are 10 instances to be implemented in this chapter. In these 10 instances, 2 instances were taken from Senyu & Toyada (1967) (it will be called as SENTO instances), and 8 instances were taken from Weingartner & Ness (1967) (it will be called as WEING instances). Table 5 shows the problem characteristics for all instances to be implemented.

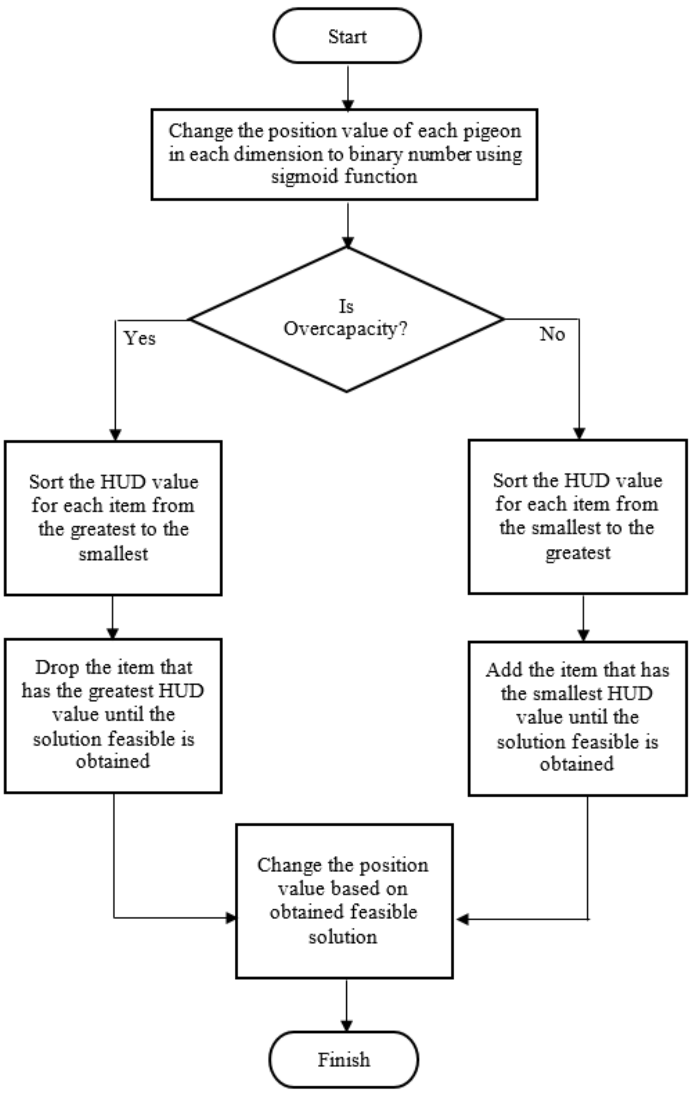
Start
Change the position value of each pigeon in each dimension to binary number using sigmoid function
Is Overcapacity?
Yes
No
Sort the HUD value for each item from the greatest to the smallest
Sort the HUD value for each item from the smallest to the greatest
Drop the item that has the greatest HUD value until the solution feasible is obtained
Add the item that has the smallest HUD value until the solution feasible is obtained
Change the position value based on obtained feasible solution
Finish

Figure 1. Drop/add items procedure

```
1.  Input the data of MKP problem: the number of knapsacks, the number of items, the
    weight of each item, capacity of each knapsack, benefits of each item.
2.  Input the value of parameters that are used in PIO: the number of iterations for map and
    compass operator (Nc1max), the number of iterations for landmark operator (Nc2max),
    the number of pigeons (Np), the number of dimensions (D), the value of map and
    compass operator factor (R).
3.  Calculate $HUD_i$ using equation (12).
4.  Initialize Vpd0 by generating the random number between 0 and 1.
5.  Initialize Xpd0 by generating the random number between 0 and 1.
6.  Do drop/add item procedure.
7.  Calculate the fitness value, fitnessB, fitnessg, and update XB and Xg.
8.  Do map and compass operator calculation.
9.  If the number of iterations has not yet reached Nc1max or other predetermined
    condition, back to step 8; otherwise, go to step 10.
10. Do landmark operator calculation.
11. If the number of iterations has not yet reached Nc2max or other predetermined
    condition, back to step 10, otherwise is finished.
```

Figure 2. Pseudocode of PIO for MKP

4.2. Parameter setting of PIO

Before implementing PIO on the instances, we determine the best parameter value of PIO to be used in this research. The determination of the PIO parameter will be done using one factor at a time (OFAT). The initial value used in this research is taken from the parameter value from the initial paper on PIO by Duan and Qiao in 2014). The initial value of these parameters are Nc1max = 150, Nc2max = 8, Np = 150 and R = 0.2. The application of OFAT in this paper is done by five replications. There are only two cases that will be experimented with to get the best parameter, namely SENTO1 and WEING7 cases, because those two cases are the most complex than others. The first parameter to be tested is NC1max. In the experiment, the value of Nc1max will be subtracted and added by a multiple of 100. In running the program, there are 21 levels or treatments for changing the value of the Nc1max parameter. Table 6 and 7 below shows the combination of parameters for changing the value of the Nc1max parameter in the SENTO1 case and the result from this parameter combination.

Based on table 11 above, the Nc1max parameter of 1450 yields the highest performance, so the best value of the Nc1max parameter is 1450 for SENTO 1 case. This NC1max best value then will be used on the determination of the remaining other parameters. After the best value of the Nc1max parameter is obtained, then the best parameter value for the other PIO parameter will be determined using the same fashion. Np parameter value will also be subtracted and added by a multiple of 100. Nc2max values in this experiment are in the range of 1 to 9. R parameter values in this experiment are in the range of 0.1 to 1, with an increase of 0.1 in every treatment.

Based on these experiments, we get that the best parameter combination value of the PIO parameter for SENTO1 case are Nc1max = 1450, Nc2max = 6, Np = 450 and R = 0.3. After the best parameter combination value of the PIO parameter for SENTO 1 has been determined, we determine the best parameter combination value of the PIO parameter for WEING 7 in the same way that we have done in SENTO 1. The best parameter combination value of the PIO parameter for WEING7 case are Nc1max = 1150, Nc2max = 4, Np = 300 and R = 0.8.

4.3. Numerical experiment

Based on the parameter setting above, it is obtained two combinations of a parameter that yield the best performance of PIO. The first combination is Nc1max = 1450, Nc2max = 6, Np = 450 and R = 0.3 and the second combination is Nc1max = 1150, Nc2max = 4, Np = 300 and R = 0.8. Both combinations will be applied to all 10 cases in this research. The result of PIO implementation for all ten instances using the first parameter combination is shown in Table 8 below.

Table 9 below shows the result of PIO implementation for all ten instances using the second parameter combination.

After both parameters, combinations are used in PIO implementation for all ten instances, and the result is obtained, we will choose the best value from one of two parameter combinations. Table 10 below shows the summary of the implementation of two-parameter combinations and the best value from one of the two-parameter combinations.

Performance comparison is made by comparing the best objective function on all ten instances with the result obtained by other metaheuristics. As stated before, the other metaheuristic that will be compared with PIO are particle swarm optimization (PSO), intelligent water drops (IWD), and genetic algorithm (GA). Table 11 and 12 below shows the performance comparison of PIO, PSO, IWD, and GA in all ten instances.

As we see in Table 11, the best known is the best value that has ever known to date. IWD is not applied to SENTO1 and SENTO2 instances, while GA is not applied to WEING1, WEING2, WEING3, WEING4, WEING5, and WEING6 instances. PSO is applied to all ten instances. From the result above, it can be shown that the performance of PIO is a satisfactory result compared to all the compared metaheuristics.

5. CONCLUSIONS

Pigeon-inspired optimization (PIO) can be applied to Multidimensional Knapsack Problem (MKP) using a random number coding method for initialization and binary number from sigmoid functions for the decoding process. In this encoding and decoding process, HUD is used, which is one of the simplest local heuristic methods to manage the constraints of the MKP. From the coding method and the decode is then translated into the position of the pigeon on the map and the compass operator and the landmark operator, from which the position value will be calculated the objective function value of fitness as well as the global position of the global pigeon which determines which items will be chosen into a knapsack.

In the implementation of the PIO algorithm to the ten instances, it is known that overall the performance of the PIO algorithm is better than the performance of the PSO algorithm and the performance of the GA algorithm in its application to MKP. However, this PIO algorithm gets slightly worse results than the IWD algorithm in its application to MKP. For further research, it is possible to apply the PIO algorithm to solve other discrete problems and combinatorial problems.

Table 6: Parameter combination by changing NC1max parameter on SENTO1 case

Parameter combination	Nc1max	Nc2max	Np	R
1	50	8	150	0,2
2	150	8	150	0,2
3	250	8	150	0,2
4	350	8	150	0,2
5	450	8	150	0,2
6	550	8	150	0,2
7	650	8	150	0,2
8	750	8	150	0,2
9	850	8	150	0,2
10	950	8	150	0,2
11	1050	8	150	0,2
12	1150	8	150	0,2
13	1250	8	150	0,2
14	1350	8	150	0,2
15	1450	8	150	0,2
16	1550	8	150	0,2
17	1650	8	150	0,2
18	1750	8	150	0,2
19	1850	8	150	0,2
20	1950	8	150	0,2
21	2050	8	150	0,2

Table 7: The result of parameter combination by changing NC1max parameter on SENTO1 case

Parameter combination	Replication					Best	Average	STDEV
	1	2	3	4	5			
1	7566	7586	7598	7586	7588	7598	7584.8	11.63
2	7617	7622	7642	7641	7576	7642	7619.6	26.80
3	7623	7643	7598	7598	7598	7643	7612	20.43
4	7654	7622	7632	7643	7637	7654	7637.6	11.97
5	7598	7639	7598	7622	7625	7639	7616.4	17.98
6	7654	7598	7598	7615	7599	7654	7612.8	24.14
7	7654	7641	7598	7635	7643	7654	7634.2	21.37
8	7598	7654	7636	7632	7642	7654	7632.4	20.95
9	7664	7599	7598	7674	7625	7674	7632	35.64
10	7624	7637	7626	7625	7625	7637	7627.4	5.41
11	7637	7660	7642	7651	7654	7660	7648.8	9.26
12	7636	7632	7598	7635	7692	7692	7638.6	33.78
13	7598	7598	7625	7679	7679	7679	7635.8	40.95
14	7636	7626	7606	7679	7692	7692	7647.8	36.36
15	7642	7642	7636	7625	7719	7719	7652.8	37.65
16	7637	7622	7679	7636	7654	7679	7645.6	21.85
17	7632	7637	7664	7632	7635	7664	7640	13.58
18	7654	7637	7636	7634	7598	7654	7631.8	20.52
19	7643	7599	7625	7637	7635	7643	7627.8	17.36
20	7598	7651	7684	7642	7609	7684	7636.8	34.40
21	7692	7632	7692	7598	7654	7692	7653.6	40.33

Table 8: PIO Implementation using first parameter combination

Instances	Replication					
	1	2	3	4	5	Best
SENTO1	7679	7679	7719	7679	7679	7719
SENTO2	8619	8653	8645	8637	8613	8653
WEING1	141148	141278	141148	141148	141258	141278
WEING2	130103	130723	130723	130123	130163	130723
WEING3	95677	95007	95007	95467	95137	95677
WEING4	119337	119337	119337	119337	119337	119337
WEING5	98396	98396	98495	98396	98396	98495
WEING6	130123	130113	130113	130113	130113	130123
WEING7	1094452	1094417	1094356	1094547	1095007	1095007
WEING8	612430	614706	611555	612737	615315	615315

Table 9. PIO Implementation using second parameter combination

Instances	Replication					
	1	2	3	4	5	Best
SENTO1	7719	7679	7679	7679	7679	7719
SENTO2	8649	8619	8618	8599	8665	8665
WEING1	141148	141258	141148	141148	141148	141258
WEING2	130103	130712	130103	130883	130103	130883
WEING3	95007	95007	95007	95007	95007	95007
WEING4	119337	119337	119337	119337	119337	119337
WEING5	98495	98396	98396	98396	98396	98495
WEING6	130623	130202	130123	130113	130123	130623
WEING7	1094262	1094811	1094467	1094111	1094356	1094811
WEING8	612880	611100	617901	614553	613751	617901

Table 10: PIO Implementation using both parameter combination

Instances	Best value of two-parameter combinations		Best
	1	2	
SENTO1	7719	7719	7719
SENTO2	8653	8665	8665
WEING1	141278	141258	141278
WEING2	130723	130883	130883
WEING3	95677	95007	95677
WEING4	119337	119337	119337
WEING5	98495	98495	98495
WEING6	130123	130623	130623
WEING7	1095007	1094811	1095007
WEING8	615315	617901	617901

Table 11: Performance comparison of PIO, PSO, IWD, GA in all ten instances

Instances	PSO	IWD	GA	PIO	Best Known
SENTO1	7,725	-	7,626	7,719	7,772
SENTO2	8,716	-	8,685	8,665	8,722
WEING1	138,927	141,278	-	141,278	141,278
WEING2	125,453	130,883	-	130,883	130,883
WEING3	92,297	95,677	-	95,677	95,677
WEING4	116,622	119,337	-	119,337	119,337
WEING5	93,678	98,796	-	98,495	98,796
WEING6	128,093	130,623	-	130,623	130,623
WEING7	1,059,560	1,094,736	1,093,987	1,095,007	1,095,445
WEING8	492,347	620,872	613,383	617,901	624,319

Table 12: Performance comparison of PIO, PSO, IWD, GA in percentage to best known

Instances	PSO	IWD	GA	PIO
SENTO1	99.40%	-	98.12%	99.32%
SENTO2	99.93%	-	99.58%	99.35%
WEING1	98.34%	100.00%	-	100.00%
WEING2	95.85%	100.00%	-	100.00%
WEING3	96.47%	100.00%	-	100.00%
WEING4	97.72%	100.00%	-	100.00%
WEING5	94.82%	100.00%	-	99.70%
WEING6	98.06%	100.00%	-	100.00%
WEING7	96.72%	99.94%	99.87%	99.96%
WEING8	78.86%	99.45%	98.25%	98.97%

REFERENCES

Balev, S., Yanev, N., Freville, A., Andonov, R. (2008). A Dynamic Programming Based Reduction Procedure for the Multidimensional 0-1 Knapsack Problem. *European Journal of Operational Research*, 186 (1), 63-76.

Banati, H. & Bajaj, M. (2011). Firefly Based Feature Selection Approach. *International Journal of Computer Science Issues*, 8(4), 473-480.

Basset, M.A., Doaa, E.-S., Faris, H., Mirjalili, S. (2019). A Binary Multi-Verse Optimizer for 0-1 Multi-dimensional Knapsack Problems with Application in Interactive Multimedia Systems. *Computers & Industrial Engineering*, 132, 187-206.

Basset, M.A., El-Shahat, D., El-Henawy, I., Sangaiah, A.K. (2018). A Modified Flower Pollination Algorithm for the Multi-dimensional Knapsack Problem: Human-Centric Decision Making. *Soft Computing*, 22, 4221-4239.

Battiti, R. & Tecchiolli, G. (1994). The reactive tabu search. *ORSA Journal on Computing*, 6, 126-140.

Berberler, M.E., Guler, A., Nuriyev, U.G. (2013). A Genetic Algorithm to Solve the Multi-dimensional Knapsack Problem. *Mathematical and Computational Applications*, 18(3), 486-494.

Bertsimas, D. & Demir, R. (2002). An Approximate Dynamic Programming Approach to Multidimensional Knapsack Problem. *Management Science*, 48, 550-565.

Bolaji, A.L., Babatunde, B.S., Shola, P.B. (2017). Adaptation of Binary Pigeon-Inspired Algorithm for Solving Multidimensional Knapsack Problem. *Soft Computing: Theories and Applications*, 743-751.

Bolaji, A.L., Okwonu, F.Z., Shola, P.B., Balogun, B.S., Adubisi, O.D. (2020). A Modified Binary Pigeon-Inspired Algorithm for Solving the Multi-dimensional Knapsack Problem. *Journal of Intelligent Systems*, 30(1), 90-103.

Chen, W.-N., Zhang, J., Chung, H.S.H., Zhong, W.-L., Wu, W.-G., Shi, Y.-H. (2010). A Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization Problems. *IEEE Transactions on Evolutionary Computation*, 14(2), 278-300.

Chih, M. (2015). Self-Adaptive Check and Repair Operator-Based Particle Swarm Optimization for the Multidimensional Knapsack Problem. *Applied Soft Computing*, 26, 378-389.

Chu, P.C. & Beasley, J.E. (1998). A Genetic Algorithm for the Multidimensional Knapsack Problem. *Journal of Heuristics*, 4, 63-86.

Dammeyer, F. & Voβ, S. (1993). Dynamic Tabu List Management Using the Reverse Elimination Method. *Annals of Operations Research*, 41(2), 29-46.

Deng, Y. & Duan H. (2016). Control Parameter Design for Automatic Carrier Landing System via Pigeon-Inspired Optimization. *Nonlinear Dynamics*, 85, 97-106.

Dou, R. & Duan, H. (2016). Pigeon Inspired Optimization Approach to Model Prediction Control for Unmanned Air Vehicles. *Aircraft Engineering and Aerospace Technology: An International Journal*, 88 (1), 108-116.

Duan, H. & Li, C. (2014). Target Detection Approach for UAVs via Improved Pigeon-Inspired Optimization and Edge Potential Function. *Aerospace Science and Technology*, 39, 352-360.

Duan, H. & Qiao, P. (2014). Pigeon Inspired Optimization: A New Swarm Intelligence Optimizer for Air Robot Planning. *International Journal of Intelligent Computing and Cybernetics*, 7, 24-37.

Fingler, H., Caceres, E.N., Mongelli, H., Song, S.W. (2014). A Cuda Based Solution to the Multidimensional Knapsack Problem Using the Ant Colony Optimization. *Procedia Computer Science*, 29, 84-94.

Freville, A. & Plateau, G. (1986). Heuristics and Reduction Methods for Multiple Constraints 0-1 Linear Programming Problems. *European Journal of Operational Research*, 24, 206-215.

Gilmore, P.C. & Gomory, R. (1966). The Theory and Computation of Knapsack Functions. *Operations Research*, 14(6), 1045-1074.

Glover, F. & Kochenberger, G.A. (1996). Critical Event Tabu Search for Multidimensional Knapsack Problem. *Meta-heuristics*, Springer, 407-202.

Haddar, B., Khemakhem, M., Hanafi, S., Wilbaut, C. (2016). A Hybrid Quantum Particle Swarm Optimization for the Multidimensional Knapsack Problem. *Engineering Applications of Artificial Intelligence*, 55, 1-13.

Hembecker, F., Lopes, H.S., & Godoy Jr., W. (2007). Particle Swarm Optimization for the Multidimensional Knapsack Problem. *Proceedings of the Eighth International Conference on Adaptive and Natural Computing Algorithms*, 4431, 358-365.

Ke, L., Feng, Z., Ren, Z., Wei, X. (2010). An Ant Colony Optimization Approach for the Multidimensional Knapsack Problem. *Journal of Heuristics*, 16(1), 65-83.

Khuri, S., Back, T., & Heitkotter, J. (1994). The Zero/One Multiple Knapsack Problem and Genetic Algorithms. *Proceedings of the ACM Symposium on Applied Computing* (SAC'94).

Kong, M., & Tian, P. (2006). Apply the Particle Swarm Optimization to the Multi-dimensional Knapsack Problem, In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L., Zurada, J. (Eds.), Artificial Intelligence and Soft Computing–ICAISC 2006, Vol 4029 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 1140-1149.

Kong, M., Tian, P., Kao, Y. (2008). A New Ant Colony Optimization Algorithm for the Multidimensional Knapsack Problem. *International Journal of Applied Metaheuristic Computing*, 3(4), 43-63.

Ktari, R. & Chabchoub, H. (2013). Essential Particle Swarm Optimization Queen with Tabu Search for MKP Resolution. *Computing*, 95(9), 897-921.

Lai, X., Hao, J.-K., Glover, F., Lu, Z. (2018). A Two Phase Tabu Evolutionary Algorithm for the 0-1 Multi-dimensional Knapsack Problem. *Information Sciences*, 436-437.

Leung, S.C., Zhang, D., Zhou, C., Wu, T. (2012). A Hybrid Simulated Annealing Meta-heuristic algorithm for the Two-Dimensional Knapsack Packing Problem. *Computer and Operations Research*, 39(1), 64-73.

Manzini, R., Speranza, M.G. (2012). Coral: An Exact Algorithm for the Multi-dimensional Knapsack Problem. *INFORMS Journal of Computing*, 24(3), 399-415.

Martello, S. & Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. Chicester: John Wiley & Sons Ltd.

Martins, J.P., Longo, H., Delbern, A.C. (2014). On the Effectiveness of Generic Algorithms for the Multidimensional Knapsack Problem. *Proceedings of the 2014 conference companion on Genetic and evolutionary computation companion*, ACM, 73-74.

Palit, S., Sinha, S.N., Molla, M.A., Khanra, A., Kule, M. (2011). A Cryptanalytic Attack on the Knapsack Cryptosystem using Binary Firefly Algorithm. *Second International Conference on Computer and*

Communication Technology, 428-432.

Rezoug, A., Boughaci, D., Rezoug, A. (2015). Stochastic Local Search Combined with Simulated Annealing for the 0=1 Multi-dimensional Knapsack Problem. *Symposium on Complex Systems and Intelligent Computing* (CompSIC).

Senyu, S. & Toyada, Y. (1967). An Approach to Linear Programming With 0-1 Variables. *Management Science*, 15(4), 196-207.

Shah-Hosseini, H. (2009). The Intelligent Water Drops Algorithm: A Nature-inspired Swarm-based Optimization Algorithm. *International Journal of Bio-Inspired Computation*, 1, 71-79.

Shih, W. (1979). A Branch and Bound Method for the Multiconstraint Zero-one Knapsack Problem. *Journal of the Operational Research Society*, 15, 196-207.

Tisna, A., F.D., Abusini, S., Andar, A. (2013). Hybrid Greedy-Particle Swarm Optimization-Genetic Algorithm and Its Convergence to Solve Multidimensional Knapsack Problem 0-1. *Journal of Theoritical and Applied Information Technology*, 58(3), 522-528.

Vasquez, M. & Hao, J.-K. (2001). Une Approche hybride pur le sac a dos multi-dimensionnel en variales 0-1. *Operations Research*, 35(4), 415-438.

Vasquez, M. & Vimont, Y. (2005) Improved Results on the 0-1 Multi-dimensional Knapsack Problem. *European Journal of Operational Research*, 165(1), 70-81.

Vimont, Y., Boussier, S., Vasquez, M. (2008). Reduced Cost Propagation in an Efficient Impicit Enumeration for the 0-1 Multi-dimensional Knapsack Problem. *Journal of Combinatorial Optimization*, 15(2), 165-178.

Wan, N.F. & Nolle, L. (2009). Solving a Multi-dimensional Knapsack Problem Using a Hybrid Particle Swarm Optimization Algorithm. *Proceedings of the 23rd European Conference on Modelling and Simulation*, ECMS 2009.

Weingartner, H.M., & Ness, D.N. (1967). Methods for the Solution of the Multi-dimensional 0/1 Knapsack Problem. *Operation Research*, 15(1), 83-103.

Zhang, X., Wu, C., Li, J., Wang, X., Yang, Z., Lee, J.-M., Jung K.-H. (2016). Binary Artificial Algae Algorithm for Multidimensional Knapsack Problem. *Applied Soft Computing*, 43, 583-595.

This page is intentionally left blank