Academic Platform Journal of Engineering and Science

journal homepage: http://apjes.com/

An ABC Algorithm Inspired by Boolean Operators for Knapsack and Lot Sizing Problems

*[1]Emrah Hancer(0000-0002-3213-519)

[1]Mehmet Akif Ersoy University, Department of Computer Technology and Information Systems, Burdur, Turkey

ehancer@mehmetakif.edu.tr

Received Date: 10.09.2017 Accepted Date: 31.05.2018

Abstract.

This paper proposes a logically inspired artificial bee colony algorithm (ABCLO) to deal with the knapsack and lot sizing problems shown in many forms such as in economics, engineering and business. The proposed ABC-LO algorithm aims to find fitter solutions using the search mechanism designed through the basic Boolean operators. To verify the effectiveness of the ABC-LO algorithm, it is analyzed and compared with the recent variants of particle swarm optimization, artificial bee colony and genetic algorithms. The results indicate that the proposed ABC-LO algorithm performs well in knapsack and lot sizing problem sets compared to the others.

***Keywords**:* artificial bee colony, Boolean logic, knapsack, lot sizing.

1 INTRODUCTION

In recent years, swarm intelligence based algorithms have attracted attention due to the ability of producing promising solutions in a reasonable time. Among swarm intelligence based algorithms, artificial bee colony (ABC) [1] is one of the most robust, recent and popular algorithms proposed to solve real-parameter, non-convex and non-smooth problems. The standard ABC algorithm synergizes minimalistic foraging procedure with waggle dance mechanism and is enacted through a bee colony equally partitioned among employed and onlooker bees [2].

The neighborhood search is carried out through positional perturbation of the foragers in search for fitter food sources followed by the greedy selection in order to keep the positions of the fitter source. The waggle dance mechanism is performed by employed bees to share information with onlooker bees through a kind of fitness and roulette-wheel based selection mechanism. Using this information, onlooker bees tend to search in the neighborhood of fitter solutions. In addition to onlooker and employed bees in the hive, scout bees derived from employed bees are responsible of finding new sources instead of poor ones.

Since the standard ABC algorithm was first proposed to solve numeric problems, it needs to be redesigned to deal with discrete or binary problems. To apply ABC to binary problems, a number of ABC variants were developed by researchers. Kashan et al. [3] introduced a discrete ABC (DisABC) variant for uncapacited facility location problem (UFLP). In DisABC, the neighborhood search is performed by measuring the dissimilarity/similarity between binary vectors. Although DisABC performs better than binary particle swarm optimization (BPSO), it does not perform well in high dimensional problems. Kiran and Gunduz [4] embedded XOR logical operator into the basic ABC search mechanism (XORABC) to address UFLP. The experimental results showed that it performed better than BPSO and DisABC. However, XOR-ABC may converge to local minima due to the search scheme. Pampara and Engelbrecht [5] proposed an angle modulation based ABC algorithm. Despite its simplicity, transformation mechanism between the continuous and binary spaces may lead to local convergence problems. Ozturk et al. [6] proposed an improved version of DisABC (IDisABC) to automatically determine the number of clusters in the data. In contrast to DisABC, IDisABC considers all similarity cases to produce effective solutions. In another study, Ozturk et al. [7] proposed a genetically inspired ABC algorithm (GBABC) to automatically evolve clusters in the data. It was also tested on numeric and knapsack problems. It is simple implement and can search the possible solution space thoroughly but may be computationally intensive for high dimensional problems. Hancer et al. [8] proposed an advanced similarity based ABC (MDisABC) algorithm for feature selection. The

*Corresponding Author: Mehmet Akif Ersoy University, Department of Computer Technology and Information Systems, Burdur, Turkey, ehancer@mehmetakif.edu.tr

Doi: 10.21541/apjes.337415

results showed that MDisABC selected a smaller number of features and obtained higher classification accuracy than the existing approaches. It can be inferred from the above discussed studies that the potential of ABC for discrete and binary problems has not been fully investigated and the need for the studies concerning discrete ABC variants has not come to end.

The main goal of this study is to develop an ABC variant for the knapsack and lot sizing problems with the expectation of loading the knapsack with valuable items and minimizing the order cost for a company. In order to achieve this goal, the idea of using logic operations is integrated to the ABC search mechanism, referred as ABC-LO. The effectiveness of the proposed ABC-LO algorithm is verified and examined by comparing it with recent discrete evolutionary algorithms. Specifically, we will investigate:

- the performance of ABC-LO versus the existing approaches on knapsack problem sets,
- the performance of ABC-LO versus the existing approaches on uncapacited lot sizing problem sets, and
- the performance of ABC-LO versus the existing approaches on capacited lot sizing problem sets.

The rest of the paper is organized as follows. Section 2 defines the considered problems and provides a general knowledge concerning Boolean operators. Section 3 presents the proposed ABC algorithm with its implementations. Section 4 introduces the experimental design and Section 5 presents the experimental results. Finally, Section 6 concludes the study through providing an insight into the future trends.

2 BACKGROUND

2.1 Knapsack Problem

Suppose that a friend living abroad wants from you to bring some devices or items. It is known that one can only have the luggage not exceeding 15-30 kilograms in the planes. Therefore, both valuable and lightweight items should be chosen to pick a luggage. Well then, how to pick a luggage in an optimal way? That question can be enhanced to the many real world problems such as in economics, industry, transportation, logistic, computer science and etc. All these problems are considered in the same structure, referred as the “knapsack problem”. The mathematical description of the knapsack problem can be presented by:

$$\max f(x) = \sum_{j=1}^{n} c_j x_j \qquad (1)$$

$$subject\ to \begin{cases} \sum_{j=1}^{n} w_j x_j \le W \\ x_j \in \{0,1\}\ where\ j = (0,1 \dots, n) \end{cases}$$

where w_j represents weight of the jth item; c_j represents cost of the jth item; x_j denotes the status of the jth item, whether it is loaded into knapsack or not (1 or 0); and W is the capacity of knapsack.

In order to solve the knapsack problem, a large number of methods such as branch-bound [10], dynamic programming [9] or hybridization of both approaches [11, 12] have been proposed. In fact, evolutionary computation techniques such as genetic algorithm [13, 14], particle swarm optimization [15, 16], differential evolution [17, 18] and artificial bee colony [19, 20] have also been applied to address the knapsack problem. More information concerning the approaches proposed to cope with the knapsack problem can be found in [10].

2.2 Lot Sizing Problem

Assume that a company orders quantities in order to provide the net requirements of customer demand. The order decision needs to be partitioned into the periods in order to minimize the total cost. But, which periods are the most optimal ones to order? Several factors such as ordering cost, holding cost, capacity, minimum-maximum order quantity, shortage cost and etc. should be considered before making decisions, i.e., it may be single or multi item, capacited or uncapacited, shortage allowed or not, and single or multi-level. In this paper, single item, single level and no shortages allowed model is considered with both capacited and uncapacited versions.

The basic notations for lot sizing are defined as follows:

K: setup cost

h: holding cost

O_i : Order quantity for period i

R_i : Net requirements for period i

I_i : Ending inventory for period i

x_i : An order decision is made or not for period i

Cap : Capacity of the order quantity

In terms of the notations, the lot sizing problem can be defined as follows:

$$\min \sum_{i=1}^{n} \left(Kx_i + hI_i \right) \quad (2)$$

subject to: $I_0 = 0$ (3)

$$I_{i-1} + x_i O_i - I_i = R_i \quad (4)$$

$$I_i \geq 0 \ , \ i \in \{1,2,...,n\} \quad (5)$$

$$O_i \geq 0 \quad (6)$$

$$x_i \in \{0,1\} , \ i \in \{1,2,...,n\} \quad (7)$$

$$O_i < Cap \quad (8)$$

where Eq. (4) means no initial inventory is available before starting trade; Eq. (5) keeps the inventory balance between requirements and orders; Eq. (6) reveals no shortage is allowed; and Eq. (8) is for the capacited form of lot sizing.

Many approaches have been proposed to deal with the lot sizing problem. Wagner and Whitin [21] introduced a method based on dynamic programming, which guarantees optimal solutions. Tasgetiren and Liang [22] proposed a modified version of PSO to solve single item, single level and uncapacited lot sizing problem. The results indicated that the modified PSO algorithm performed better than GA. Deroussi and Lemoine [23] hybridized BPSO with the Wagner-Whitin algorithm to cope with multi-level lot sizing problems. The detailed information concerning lot sizing problem solvers can be found in [24, 25].

2.3 Boolean Algebra

In Boolean algebra, there are two truth variables, known as true (1) and false (0), respectively. In contrast to elementary algebra using addition and multiplication, the main operators of Boolean algebra are as follows:

- AND (conjuction), denoted by x Λ y, satisfies 1 if x = y =1; otherwise 0.
- OR (disjunction), detoned by x ∨ y, satisfies 0 if x = y = 0; otherwise 1.
- NOT (negation), denoted by ¬ x, satisfies 1 if x = 0, and 0 if x = 1.

These Boolean operators can be combined or compounded to build other Boolean operators. The main derived operators of Boolean algebra are as follows:

- Material implication, denoted by x → y, satisfies y if x is 1.
- Exclusive or XOR, denoted by x ⊕ y, satisfies 0 if x = y = 1 or x = y = 0; otherwise 0.
- Complement of exclusive, denoted by x ≡ y, satisfies 1 if x ⊕ y = 0; otherwise 1.

3 PROPOSED ABC-LO ALGORITHM

In this section, the overall algorithm is first introduced. Then, its implementations to the knapsack and lot sizing problems are described.

3.1 Overall algorithm

Swarm intelligence (SI), which is an artificial intelligence discipline, concerns with the behaviors of natural and artificial systems composed of many individuals, including schools of fish, colonies of ants, termites flocks of birds and herds of land animals. SI based behaviours resulting from the local interactions of the individuals with each other and with their environment comprise two main characteristics: self-paced labour and self-organization. The well-accepted examples of the SI discipline are ant colony optimization (ACO) [26], particle swarm optimization (PSO) [27], bacterial foraging optimization (BFO) [28] and artificial bee colony (ABC) [1] algorithms. This paper concentrates on the ABC algorithm since it is simple, easy to implement, has fewer parameters and performs successful performance in a variety of fields such as numerical problems [29], clustering [30], image analysis [31], etc.

Although ABC has been successfully applied to several problems, the structure of ABC is not suitable to binary and discrete problems. In other words, the solution initialization and neighborhood search mechanisms of ABC need to be redesigned for binary space. In this study, binary solutions are initialized using Eq. (9) which is the most-widely preferred binary solution initialization way among binary evolutionary algorithms.

$$x_{ij} = \begin{cases} 0 & if \quad U_{ij}(0,1) < 0.5 \\ 1 & if \quad U_{ij}(0,1) > 0.5 \end{cases} \quad (9)$$

where $U_{ij}(0,1)$ is a uniformly generated number within the range of 0 and 1.

To adapt neighborhood search mechanism for the binary solution space, the idea of using Boolean operators to design search mechanism is a novel and recent way used also in XOR-ABC and XOR-PSO. Although XOR-ABC and XORPSO are the first examples of adapting search mechanism through logical operators, they suffer from the local stagnation problems arising from their design. The detailed information concerning XOR-ABC and XOR-PSO can be found in Section 4.1. Motivated from the previously

described issue, we introduce a new logically inspired ABC (ABC-LO) algorithm, in which the neighborhood search mechanism has effectively redesigned through the XOR and AND logical operators by Eq. (10).

$$V_i = X_i \oplus (\beta \otimes (X_i \oplus X_k)) \quad (10)$$

where $\oplus$ denotes XOR operator; $\otimes$ denotes AND operator; $X_i=\{x_{i1},x_{i2},\ldots,x_{iD}\}$ and $X_j=\{x_{j1},x_{j2},\ldots,x_{jD}\}$ are the current and neighbor binary solutions, respectively; and β is a randomly D-dimensional binary vector initialized by Eq. (9).

The fundamental steps of the ABC-LO algorithm are as follows:

1. Initialize the population by Eq. (9).
2. Search in the neighborhood of each solution X_i by Eq. (10).
3. Apply greedy selection between X_i and V_i.
4. Produce probabilistic value p_i for each solution X_i using roulette-wheel mechanism by Eq. (11).
5. Select *SN* number of solutions in a probabilistic manner, where $rand(0,1) > p_i$.
6. Search in the neighborhood of each selected solution by Eq. (10).
7. Apply greedy selection between the selected solution and its neighborhood.
8. If there exists any abandoned solution which cannot be improved for a predefined number of trials, referred as "limit", a new solution is initialized by Eq. (9) instead of the abandoned one.
9. Repeat Steps 2 to 8 until the maximum number of cycles is met.

$$p_i = \frac{fitness_i}{\sum_{i=1}^{SN} fitness_i} \quad (11)$$

where *SN* is the population size and $fitness_i$ is the fitness value of X_i.

3.2 Implementations of ABC-LO

The implementations of ABC-LO are presented as follows.

1. Knapsack: Considering the implementation of the ABC-LO algorithm to the knapsack problem, each solution $X_i=\{x_{i1},x_{i2},\ldots,x_{iD}\}$ represents the probable items for the knapsack. If any position of X_i is equal to 1, the corresponding item is loaded into the knapsack. The objective function of knapsack can be defined through Eq. (1) as follows:

$$f(x_j) = \sum_{i=1}^{D} c_i x_{ji} + Q \min\left\{0, W - \sum_{i=1}^{D} w_j x_{ji}\right\} \quad (12)$$

where Q is a penalty factor.

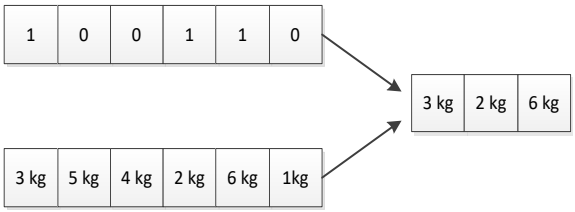

Fig. 1 An illustrative example of how items are selected

We provide the following example to show how the fitness value of a solution is evaluated in the knapsack problem. Assume that X_i = {1, 0, 0, 1, 1, 0}, w_i = {3kg, 5kg, 4kg, 2kg, 6kg, 1kg}, c_i = {2€, 1€, 4€, 2€, 5€, 6€} and W = 15kg. The items whose corresponding positions are equal to 1 are selected to load the knapsack, shown in Fig. 1. Accordingly, the total weight of items is 11 kg that does not exceed the capacity, and the total cost of items is 9€. If the total weight of items in a solution exceeds the knapsack capacity, a new solution is generated by Eq. (9) instead of that solution.

2. Lot sizing: In terms of the lot sizing problem, each solution $X_i=\{x_{i1},x_{i2},\ldots,x_{iD}\}$ represents the time periods for order decisions. If any position of X_i is equal to 1, the order is given for the corresponding period. The objective function of lot sizing is defined by Eq. (3).

We provide the following example to show how the fitness value of a solution is evaluated in the uncapacited lot sizing problem. Assume that X_i = {1,0,0,1,1,0,0}, R = {100,40,50,60,70,30,50}, setup cost (*K*) is set to 100€ and hosting cost is set to 1€. For the first period, the order quantity (O_{i1}) is set to 190 which is equal to the sum of net requirements (R_{i1}, R_{i2}, R_{i3}), since no order decision is made for the second and third periods. Then, the setup cost (Kx_{i1}) is 100€ and the remaining number of orders is 100. For the second period, the remaining number of orders is 40 and the holding cost for the requirements is 60€. For the third period, there is no remaining order, and the holding cost for the requirements is 100€ (50 x 2) since two periods were passed. For the fourth period, the order quantity (O_{i4}) is set to 60, and so the setup cost is 100€. For the fifth period, the order quantity (O_{i5}) is set to 150, and so the setup cost (Kx_{i5}) is 100€, and the number of remaining orders are 80. For the sixth period, the remaining number of orders is 50 and the holding cost for the requirements is 30€. For the final period, the holding cost for the requirements is (50x2) 100€. Then, the fitness value of X_i is calculated as 570. To clearly illustrate the example, please see Fig. 2.

	1	2	3	4	5	6	7	Total
R_{id}	100	40	50	60	70	30	50	
x_{id}	1	0	0	1	1	0	0	

O_{id}	190			60	150			
I_{id}	90			0	80			
Kx_i	100			100	100			
$I_{id}h$		40	100			30	100	
Sum (Kx_i + $I_{id}h$)								
	100	40	100	100	100	30	100	570

Fig. 2 An illustrative example for uncapacited lot sizing.

4 EXPERIMENTAL DESIGN

4.1 Algorithms Used For Comparisons

In this section, the binary evolutionary algorithms used for comparisons are presented as follows:

1. Binary particle swarm optimization (BPSO) [32]: In BPSO, each velocity is first updated by Eq. (13) as in standard PSO. For each position of velocities, a value in the range of [0,1] is generated using a sigmoid function by Eq. (14). Then, new positions of particles are determined by Eq. (15). Although it is simple to implement, it has local stagnation problems.

$$V_{ij}^{t+1} = wV_{ij}^{t} + c_1 r_1 (pbest_{ij}^{t} - x_{ij}^{t}) + c_2 r_2 (gbest_{j}^{t} - x_{ij}^{t}) \quad (13)$$

$$sig(v) = \frac{1}{1+e^{-v}} \quad (14)$$

$$x_{i,j}(t+1) = \begin{cases} 0 & rand(0,1) \geq sig(v_{i,j}(t+1)) \\ 1 & rand(0,1) < sig(v_{i,j}(t+1)) \end{cases} \quad (15)$$

where w is a inertia weight; c_1 and c_2 are balance factors between global best and local best positions; $pbest_{ij}^{t}$ is the jth position of the ith local best particle at iteration t; $gbest_{j}^{t}$ is the jth position of global best particle at iteration t; r_1 and r_2 are random numbers in the range of [0,1]; x_{ij} is the jth position of the ith particle; and *rand(0,1)* is a uniformly generated number between [0,1].

2. XOR based particle swarm optimization (XOR-PSO) [33]: In XOR-PSO, the positions of velocities and particles are updated using logic operators by Eqs. (16) and (17):

$$V_{ij}^{t+1} = w_1 \otimes (pbest_{j}^{t} \oplus x_{ij}^{t}) + w_2 \otimes (gbest_{j}^{t} \oplus x_{ij}^{t}) \quad (16)$$

$$X_{ij}^{t+1} = X_{ij}^{t} \oplus V_{ij}^{t} \quad (17)$$

where + denotes OR operator; $\otimes$ denotes AND operator; $\oplus$ denotes XOR operator; and r_1 and r_2 are uniformly generated numbers between 0 and 1.

3. XOR based Artificial Bee Colony (XOR-ABC) [4]: Inspired by XOR-PSO, the neighbor search mechanism of standard ABC is modified using XOR logical operator by Eq. (18):

$$v_{i,j} = x_{i,j} \oplus [\gamma(x_{i,j} \oplus x_{k,j})], \text{ i}\neq\text{k} \quad (18)$$

where j is the randomly selected position; $\oplus$ represents XOR operator; and γ is a uniformly generated number between 0 and 1, which behaves like a probabilistic logic gate. If γ is smaller than 0.5, the logic gate is invoked, i.e., the value of $x_{ij} \oplus x_{kj}$ will be inverted. Otherwise, the result will not be inverted.

4. Discrete Artificial Bee Colony (DisABC) [3]: DisABC uses Jaccard Coefficient to measure the similarity between binary vectors. Let, $X_i=\{x_{i1},x_{i2},\ldots,x_{iD}\}$ and $X_k=\{x_{k1},x_{k2},\ldots,x_{kD}\}$ are two binary vectors. The similarity between them is evaluated by Eq. (19), and the dissimilarity between them is then evaluated by Eq. (20). After the evaluation of dissimilarities, new solution generator mechanism is applied to find a better solution.

$$Similarity(X_i, X_k) = \frac{M_{11}}{M_{01} + M_{10} + M_{11}} \quad (19)$$

where M_{11} is the number of bits where both vector positions are equal to 1; M_{10} is the number of bits where the positions of X_i and X_k are equal to 1 and 0, respectively; M_{01} is the number of bits where the positions of X_i and X_k are equal to 0 and 1, respectively.

$$Dissimilarity(X_i, X_k) = 1 - Similarity(X_i, X_k) \quad (20)$$

5. Angle Modulated Artificial Bee Colony (AMABC) [5]: In contrast to typical binary evolutionary algorithms, AMABC searches in the four-dimensional continuous solution space, where each solution X_i is represented by (a_i, b_i, c_i, d_i), a, b, c, $d \in$ [-1,1]. To evolve solutions, the standard ABC framework is carried out. To evaluate the fitness value of a solution, that solution is mapped from the four-dimensional continuous space to the D-dimensional binary space through function $g(x)$, defined by Eq. (21):

$$g(x) = \sin(2\pi(x-a) \times b \times cos(2\pi(x-a) \times c)) + d \quad (21)$$

where a determines the amplitude of function g; b determines the frequency or period of sinus function in g; c controls the frequency or period of cosin function in g; and d controls the vertical shift.

How the mapping process is carried out is described as follows. First, the values of (a_i, b_i, c_i, d_i) are substituted into $g(x)$. Then, a D-dimensional continuous vector is generated by sampling $g(x)$ for the predefined range of x (e.g. x = {0, 0.1, 0.2,…, D×0.1}). Using this continuous vector, the D-dimensional binary solution (bitstring) is then determined by Eq. (22):

$$bitstring_{ij} = \begin{cases} 1 & vector_{ij} \geq 0 \\ 0 & otherwise \end{cases} \quad (22)$$

where $vector_i$ is a continuous vector generated by sampling $g(x)$ for the ith solution.

6. Genetic Algorithm (GA) [34]: GA belongs to the subclass of evolutionary algorithms of evolutionary computation techniques. GA has three fundamental operators: 1) selection: a number of solutions from the population are selected during each generation. Fitter solutions tend to be selected since fitness-based selection strategy is applied; 2) crossover: a number of new solutions to form new generation are produced using the selected individuals; and 3) mutation is applied to the new generation obtained by selection and mutation to ensure diversity within the population.

We should notify that the dissimilarity between any pairs of solutions in DisABC cannot be properly measured in a lot sizing problem set, since the first position of a solution should be assigned as 1 to evaluate the fitness. Thus, DisABC is not used in the experiments of the lot sizing problem.

4.2 Problem used for comparisons

To investigate the performance of the evolutionary algorithms on the knapsack problem, six benchmark problem sets (referred as 'K-Set') [35] comprising of a wide range of items from 40 to 750 and capacities from 400 to 20351.5 are selected, the details of which can be found in Table 1.

Table 1 Knapsack problem sets used in experiments.

	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6
# Orders	40	80	100	250	500	750
Capacity	400	600	2732	6536	13743	20351.5

To investigate the performance of the evolutionary algorithms on the lot sizing problem, five problem sets are uniformly generated, where the values of requirements are in the range of 50 and 500, and the total number of periods are 250. For the capacited form of lot sizing, the capacity (*Cap*) is set to 1300 in all problems.

4.3 Parameter Settings

In the experiments of knapsack, the following parameter values are used: the population size is chosen as 30 and the maximum number of cycles is set to 100 for all algorithms; the limit value of all ABC variants is chosen as 50; ϕ_{max} and ϕ_{min} values of DisABC are selected 0.9 and 0.5 as in [3]; the parameters of BPSO are selected as in [36]: c_1 = 2, c_2 = 2, w_{start} = 0.9, w_{end} = 0.4, V_{max} = 6; the crossover rate and mutation rate of GA are set to 0.8 and 0.2. In the experiments of lot sizing, the following parameter values are used: the population size is chosen as 50 and the maximum number of cycles is set to 1500 for all algorithms; the limit value of all ABC variants is chosen as 50 as in the experiments of knapsack; and the parameter values of BPSO and GA are selected as in the experiments of knapsack.

5 EXPERIMENTAL RESULTS

The results are presented in Table 2, 3 and 4 over the 30 independent runs in terms of best, worst, mean and standard deviation ('std'). In Tables, 'T' shows the results of the Wilcoxon Rank Sum Test at 95% of confidence, where '+' or '-' denotes that ABC-LO performs significantly better or worse than the corresponding algorithm, and '=' denotes that the results of ABC-LO and the corresponding algorithm is similar to each other, i.e., there exists no such a significant difference between them. The experimental study is considered in three subsections: 1) comparisons on knapsack problem sets, 2) comparisons on uncapacited lot sizing problem sets and 3) comparisons on capacited lot sizing problem sets.

5.1 Results of Knapsack

Presenting the results of binary variants on knapsack problem sets, Table 2 shows that ABC-LO performs significantly better than other algorithms in terms of best, mean and worst values in all cases. It can be also observed from Table 2 that the performance of XOR-ABC and XOR-PSO mimicking logical operations is low compared to ABC-LO in terms of loading the knapsack with valuable items. For instance, the item subsets for knapsacks selected by ABC-LO from all available items are at least 11% more valuable than XOR-ABC, and in some problem sets, that rate is increased from 11% to 20%. It can therefore be inferred that ABC-LO is the most well-designed algorithm using the principles of Boolean logic for knapsack problem sets.Considering the binary ABC variants except for ABC-LO, DisABC and XOR-ABC generally obtain similar item subsets in terms of the total value. On the other hand, AMABC performs better than DisABC and XOR-ABC almost in all cases. However, the performance of AMABC cannot be treated as successful in terms of loading the knapsack with fitter items, when compared to the proposed ABC-LO algorithm. Considering the performance of binary PSO variants, XOR-PSO performs better than BPSO in all problem sets except for the first one. Compared to the others except for ABC-LO, GA achieves successful results, thereby getting the second position after ABC-LO among all binary variants. It can therefore be concluded that ABC-LO achieves significantly better results not only than the logically inspired binary variants, but also the other binary variants based on a variety of principles.

5.2 Results of Uncapacited Lot Sizing

Presenting the results of binary variants on uncapacited lot sizing problem sets, Table 3 shows that ABC-LO reduces the order cost significantly better than others almost in all cases. Considering other algorithms, binary PSO variants, especially BPSO, cannot successfully minimize the order cost, i.e., the order cost obtained by PSO variants are very intensive compared to the others. For instance, BPSO and XOR-PSO subsequently obtain 40596.13 and 31266.13 in terms of the average total cost for the first problem, while the others obtain between 24000 and 25000. Such gap between binary PSO variants and the others can be also illustrated in other problem sets. Accordingly, it can be indicated that binary PSO variants are not suitable for uncapacited lot sizing problem sets. Furthermore, XOR-ABC and GA generally achieve similar order costs, but performs better than AMABC in all cases. Accordingly, XOR-ABC and GA gets the second position after ABC-LO in terms of minimizing the order cost. It can therefore be stated that ABC-LO is also the most well-designed algorithm mimicking the principles of Boolean algebra for uncapacited lot sizing problem sets.

Table 2 Results of knapsack problem sets.

		Set 1	**Set 2**	**Set 3**	**Set 4**	**Set 5**	**Set 6**
BPSO	Best	94.93	157.5	3727	8356	15761	22961
	Mean	91.91	152.1	3619.6	8041.7	15421	22461.8
	Std	2.06	2.16	66.22	118.7	157.6	221.2
	Worst	86.94	147.7	3477	7740	15134	22105
	T	+	+	+	+	+	+
XOR-PSO	Best	94.79	164.29	3964	8709	16618	23767
	Mean	90.71	155.67	3731.83	8423	16020.4	23148.6
	Std	3.04	4.49	110.1	191.6	309.9	308.3
	Worst	82.82	148.58	3537	7956	15378	22634
	T	+	+	+	+	+	+
GA	Best	95.35	166.36	4085	9340	17891	25434
	Mean	91.92	162.16	3977.27	8902.83	17341.7	24868.1
	Std	2.22	2.54	94.77	230.8	376.2	356.4
	Worst	87.07	146.76	3618	8480	16521	24095
	T	+	+	+	=	=	+
DisABC	Best	95.71	147.07	3494	7657	14803	21870
	Mean	90.96	140.04	3310.1	7373.97	14324.9	21301.9
	Std	1.37	3.89	86.19	115.6	190.8	219.6
	Worst	88.36	132.49	3149	7158	14159	20973
	T	+	+	+	+	+	+
AMABC	Best	92.93	147.7	3514	7756	15193	22134
	Mean	90.12	140.18	3340.43	7515.67	14746.7	21624.9
	Std	1.17	2.75	88.79	99.59	181.5	189.9
	Worst	88.51	136.35	3181	7288	11402	21311
	T	+	+	+	+	+	+
XOR-ABC	Best	87.7	148.76	3498	7759	14946	21919
	Mean	84.58	141.33	3327.43	7528.13	14733.8	21569.7
	Std	1.56	6.15	70.33	91.31	106.8	167.2
	Worst	81.68	136.91	3212	7358	14586	21170
	T	+	+	+	+	+	+
ABC-LO	Best	96.22	172.1	4139	9363	17895	26068
	Mean	94.63	167.1	4010.1	8979.8	17369.4	25274.6
	Std	1.26	3.22	73.2	165.6	266.5	367.1
	Worst	90.49	160.31	3875	8655	16578	24655

Table 3 Results of uncapacited lot sizing problem sets.

		L-Set 1	L-Set 2	L-Set 3	L-Set 4	L-Set 5
BPSO	Best	38729	37130	37673	37253	37574
	Mean	49596.13	39258.9	38931.7	40182.03	39672.43
	Std	996.74	1059.06	560.83	1104.61	788.99
	Worst	42505	41192	40029	42163	41399
	T	+	+	+	+	+
XOR-PSO	Best	36243	36866	35966	37082	36077
	Mean	40843.13	40030.8	39364.1	40329	40546.77
	Std	1712.74	1327.63	1480.8	1555.31	2197.36
	Worst	43989	44721	42006	43380	45381
	T	+	+	+	+	+
GA	Best	24446	24420	24318	24411	24292
	Mean	24567.27	24678.2	24548.2	24616.9	24543.27
	Std	129.59	275.41	209.62	139.51	205.74
	Worst	25047	25827	25043	24887	25120
	T	+	+	=	+	+
AMABC	Best	24842	24840	24855	24839	24825
	Mean	24899.5	24893.4	24888.5	24877.7	24862.2
	Std	16.84	17.01	15.51	19.68	21.23
	Worst	24924	24916	24911	24924	24898
	T	+	+	+	+	+
XOR-ABC	Best	24469	24443	24282	24420	24260
	Mean	24586.73	24615.07	24451.73	24645.23	24451
	Std	84.43	127.59	133.75	139.87	114.27
	Worst	24769	24883	24863	24954	24612
	T	+	+	=	+	+
ABC-LO	Best	24446	24418	24288	24388	24217
	Mean	24465.77	24549.2	24446.83	24507.73	24350.9
	Std	15.47	105.38	108.1	139.39	104.37
	Worst	24498	24823	24685	24977	24612

5.3 Results of Capacited Lot Sizing

Presenting the results of binary variants on lot sizing problem sets in terms of the capacited form, Table 4 shows that ABC-LO also performs significantly better than other binary variants in all cases. Further, the cost difference between ABC-LO and others in the capacited form becomes wider than the uncapacited form (see Table 3). For instance, the gap between ABC-LO and XOR-ABC, both of which are based on logic operators, is increased nearly from 121 to 170 for the first problem set. Such a gap can be also illustrated on different problem sets. Considering other algorithms, binary PSO variants cannot also perform well in reducing the order cost in all cases as in the uncapacited form. Furthermore, XOR-ABC and GA reduces the order cost better than AMABC, but XOR-ABC performs better than GA. Therefore, it can be concluded without no doubt that ABC-LO can better search the possible solution space yielding lower order cost not only in uncapacited form, but also in capacited form.

Table 4 Results of capacited lot sizing problem sets.

		L-Set 1	L-Set 2	L-Set 3	L-Set 4	L-Set 5
BPSO	Best	49250	45973	42915	50095	47125
	Mean	51855.38	48861.55	47993.16	53492.23	50255.4
	Std	1929.23	2039.76	2156.19	3038.88	2492.79
	Worst	54637	53774	51174	58390	54514
	T	+	+	+	+	+
XOR-PSO	Best	29382	29007	29049	29071	28676
	Mean	31266.13	30942.63	31399.1	31138.73	30672.27
	Std	978.65	1036.92	1142.62	1183.56	771.92
	Worst	33333	32463	34176	33615	32403
	T	+	+	+	+	+
GA	Best	24475	24451	24300	24395	24234
	Mean	24754.57	24640.1	24466.5	24608.47	24498.1
	Std	208.33	199.17	116.14	183.12	199.49
	Worst	25263	25329	24770	25098	25052
	T	+	+	+	+	+
AMABC	Best	24852	24836	24829	24850	24797
	Mean	24885.73	24881.73	24873.1	24881.3	24854.4
	Std	15.12	18.9	15.58	14.04	23.87
	Worst	24915	24909	24899	24904	24891
	T	+	+	+	+	+
XOR-ABC	Best	24486	24420	24282	24393	24255
	Mean	24668.73	24582.57	24423.4	24616.1	24423.83
	Std	168.41	111.27	123.08	164.98	146.34
	Worst	25109	24834	24736	25006	24812
	T	+	+	+	+	+
ABC-LO	Best	24447	24418	24282	24388	24212
	Mean	24497.7	24469.8	24348.2	24440.4	24313.97
	Std	31.15	65.68	56.97	52.87	65.37
	Worst	24579	24781	24547	24594	24515

6 CONCLUSIONS

The overall goal of this paper is to develop an ABC variant mimicking the principles of Boolean logic for the knapsack and lot sizing problems. This goal has been achieved by embedding a new locally inspired search mechanism into ABC.

To verify the effectiveness of the proposed algorithm, it was compared and examined on knapsack and lot sizing problem sets with a number of recent binary variants which are XOR-ABC, AMABC, DisABC, XOR-PSO, BPSO and GA. The results showed that ABC-LO outperformed the others in terms of all the considered problems. Accordingly, it can be inferred that ABC-LO is much better designed algorithm using the principles of Boolean logic than XOR-ABC and XOR-PSO. To our knowledge, it is the first time in the literature an evolutionary algorithm has been applied to different types of industrial problems in a study. In the future, it will be interesting if the proposed algorithm is hybridized using the principles of quantum computing, which has been recently applied to develop new evolutionary algorithms.

REFERENCES

[1] Karaboga, D., Basturk, B. 2011. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. Journal of Global Optimization, 39 (3), 459-471.

[2] Das, S., Biswas, S., Kundu S. 2013. Synergizing fitness learning with proximity-based food source selection in artificial bee colony algorithm for numerical optimization. Applied Soft Computing, 13 (12), 4676 - 4694.

[3] Kashan, M. H., Nahavandi, N., Kashan, A. H. 2012. DisABC: A new artificial bee colony algorithm for binary optimization. Applied Soft Computing, 12 (1), 342- 352.

[4] Kiran M. S., Gunduz M. 2013. XOR-based artificial bee colony algorithm for binary optimization. Turkish Journal of

Electrical Engineering & Computer Sciences, 21 (Sup.2), 2307-2328.
[5] Pampara, G., Engelbrecht, A. P. 2011. Binary artificial bee colony optimization IEEE Symposium on Swarm Intelligence (SIS), 11-15 April, Paris, 1-8
[6] Ozturk, C., Hancer, E., Karaboga, D. 2015. Dynamic Clustering with Improved Binary Artificial Bee Colony Algorithm, Applied Soft Computing, 28, 69-80.
[7] Ozturk, C., Hancer, E., Karaboga, D. 2015. A Novel Binary Artificial Bee Colony Algorithm Based on Genetic Operators, 297, 154-170.
[8] Hancer, E., Xue B., Karaboga, D., Zhang, M. 2015. A binary ABC algorithm based on advanced similarity scheme for feature selection, Applied Soft Computing,
36, 334-348.
[9] Andonov, R., Poirriez, V., Rajopadhye, S. 2000. Unbounded knapsack problem: Dynamic programming revisited. European Journal of Operational Research, 123 (2), 394-407.
[10] Martello, S., Toth, P. 1990. Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Inc. New York, NY, USA.
[11] Martello, S., Toth, P. 1999. Dynamic programming and strong bounds for the 0-1 knapsack problem. Management Science, 45 (3), 414–424.
[12] Martello, S., Toth, P. 1984. A mixture of dynamic programming and branch-and-bound for the subset-sum problem. Management Science, 30 (6), 765–771.
[13] Singh, R. P. 2011. Solving 0-1 Knapsack problem using Genetic Algorithms, 3rd IEEE International Conference on Communication Software and Networks (ICCSN), 27-29 May, Xi'an, 591-595.
[14] Chu, P. C., Beasley, J. E. 1998. A Genetic Algorithm for the Multidimensional Knapsack Problem. Journal of Heuristics, 4 (1), 63-86.
[15] AbdulHalim, M.F., Attea, B.A., Hameed, S.M. 2008. A binary Particle Swarm Optimization for attacking knapsacks Cipher Algorithm. International Conference on Computer and Communication Engineering (ICCCE), 13-15 May, Kuala Lumpur, 77-81.
[16] Fangguo, H. 2009. An Improved Particle Swarm Optimization for Knapsack Problem. International Conference on Computational Intelligence and Software Engineering (CISE), 11-13 December, Wuhan, 1-4.
[17] Changshou, D., Bingyan, Z., Yanling, Y., Anyuan, D. 2009. Modified Dynamic Differential Evolution for 0-1 Knapsack Problems. International Conference on Computational Intelligence and Software Engineering (CISE), 11-13 December, Wuhan, 1-4.
[18] Jun, S., Jian, L. 2009. Solving 0-1 Knapsack Problems via a Hybrid Differential Evolution. Third International Symposium on Intelligent Information Technology Application (IITA), 21-22 November, Nanchang, 134-137.
[19] Pulikanti, S., Singh, A. 2009. An Artificial Bee Colony Algorithm for the Quadratic Knapsack Problem. 16th International Conference on Neural Information Processing (ICONIP), December 1-5, Bangkok, 196-205.
[20] Sabet, S., Farokhi, F., Shokouhifar, M. 2012. A novel artificial bee colony algorithm for the knapsack problem. International Symposium on Innovations in Intelligent Systems and Applications (INISTA), 2-4 July, Trabzon, 1-5.
[21] Wagner, H. M., Whitin, T. M. 1958. Dynamic Version of the Economic Lot Size Model, Management Science, 50 (12), 1770-1774.
[22] Tasgetiren, M. F., Liang, Y-C. 2003. A Binary Particle Swarm Optimization Algorithm for Lot Sizing Problem. Journal of Economic and Social Research, 5 (2), 1-20.
[23] Deorussi, L., Lemoine, D. 2011. Discrete Particle Swarm Optimization for the Multi-Level Lot-Sizing Problem. Trends in Developing Metaheuristics, Algorithms, and Optimization Approaches, IGI Publishing Hershey, PA, USA.
[24] Brahimi, N., Dauzere-Peres, S., Najid, N. M., Nordli, A. 2006. Single item lot sizing problems. European Journal of Operational Research, 168 (1), 1-16.
[25] Van den Heuvel, W. 2006. The Economic Lot-Sizing Problem: New Results and Extensions. Erasmus School of Economics (ESE), Erasmus University of Rotterdam, PhD Thesis.
[26] Dorigo, M., Stutzle T. 2004. Ant Colony Optimization. Bradford Company Scituate, MA, USA.
[27] Kennedy, J., Eberhart, R. 1995. Particle swarm optimization, IEEE International Conference on Neural Networks, 1942-1948.
[28] Das, S., Biswas, A., Dasgupta, S., Abraham, A. 2009. Bacterial Foraging Optimization Algorithm: Theoretical Foundations, Analysis, and Applications pp 23-55. Abraham, A., Hassanien, A.-E., Siarry, P., Engelbrecht, A., ed. 2009. Foundations of Computational Intelligence, Springer Berlin Heidelberg.
[29] Akay, B., Karaboga, D. 2012. A modified Artificial Bee Colony algorithm for real-parameter optimization. Information Sciences, 192, 120-142.
[30] Hancer, E., Karaboga, D. 2017. A comprehensive survey of traditional, merge-split and evolutionary approaches proposed for determination of cluster number. Swarm and Evolutionary Computation, 32, 49-67.
[31] Hancer, E., Ozturk, C., Karaboga, D. 2012. Artificial Bee Colony Based Image Clustering Method. IEEE Congress on Evolutionary Computation (CEC), 10-12 June, Brisbane, 1-5.
[32] Kennedy, J., Eberhart, R. C. 1997. A discrete binary version of the particle swarm algorithm. IEEE International Conference on Computational Cybernetics and Simulation, 12-15 October, Orlando, 4104-4108.
[33] Yuan, X., Nie, H., Su, A., Wang, L., Yuan, Y. 2009. An improved binary particle swarm optimization for unit commitment problem. Expert Systems with Applications, 36 (4), 8049-8055.

[34] Holland, J. 2012. Genetic algorithms. Scholarpedia, 7, 1482.
[35] Zitzler, E., Laumans, M. 2012. Test Problems and Test Data for Multiobjective Optimizers. http://www.tik.ee.ethz.ch/sop/ (Access Date: 05/05/2016).
[36] Mirjalili S., Lewis, A. 2013. S-shaped versus V-shaped transfer functions for binary Particle Swarm Optimization. Swarm and Evolutionary Computation, 9, 1-14.