Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Improved binary artificial bee colony algorithm

Rafet DURGUT

Computer Engineering Department, Engineering Faculty, Karabuk University, Karabuk 78050, Turkey

E-mail: rafetdurgut@karabuk.edu.tr

Received May 19, 2020; Revision accepted Aug. 27, 2020; Crosschecked Aug. 4, 2021

Abstract: The artificial bee colony (ABC) algorithm is an evolutionary optimization algorithm based on swarm intelligence and inspired by the honey bees' food search behavior. Since the ABC algorithm has been developed to achieve optimal solutions by searching in the continuous search space, modification is required to apply it to binary optimization problems. In this study, we modify the ABC algorithm to solve binary optimization problems and name it the improved binary ABC (IbinABC). The proposed method consists of an update mechanism based on fitness values and the selection of different decision variables. Therefore, we aim to prevent the ABC algorithm from getting stuck in a local minimum by increasing its exploration ability. We compare the IbinABC algorithm with three variants of the ABC and other meta-heuristic algorithms in the literature. For comparison, we use the well-known OR-Library dataset containing 15 problem instances prepared for the uncapacitated facility location problem. Computational results show that the proposed algorithm is superior to the others in terms of convergence speed and robustness. The source code of the algorithm is available at https://github.com/rafetdurgut/ibinABC.

Key words: Artificial bee colony; Binary optimization; Uncapacitated facility location problem (UFLP)
https://doi.org/10.1631/FITEE.2000239 **CLC number:** TP301.6

1 Introduction

In recent years, several meta-heuristic optimization algorithms that are influenced by various phenomena of nature have been developed (Hussain et al., 2019). There are many algorithms inspired by physical, chemical, or biological phenomena and swarms of animals. An optimization algorithm is called population-based if it searches the best solution using a set of solutions (Wu et al., 2019). The population-based algorithms are divided into two evolutionary and swarm intelligence algorithms. Genetic algorithm (GA) (Holland, 1992), evolutionary strategy (Rechenberg, 1978), and differential evolution (Storn and Price, 1997) are the most popular in population-based evolutionary algorithms, while artificial bee colony (ABC) (Karaboga and Basturk, 2007), particle swarm optimization (PSO) (Kennedy and Eberhart, 1995), grey wolf optimization (GWO) (Mirjalili et al., 2014), crow search algorithm (CSA) (Askarzadeh, 2016), and whale optimization algorithm (WOA) (Mirjalili and Lewis, 2016) are the most popular for population-based swarm intelligence algorithms.

Optimization methods work on certain types of problems when they are first proposed. These optimization problems can be continuous (e.g., ABC and PSO), combinatorial (e.g., GA), binary (e.g., GA), or constrained under some conditions (e.g., CSA). An optimization algorithm can be applied to other problem types. However, by the no free lunch theorem (Mallipeddi et al., 2011), the increase in success for one problem type has the opposite effect in other problem types. Therefore, when an algorithm is applied to different types of problems, some modifications are required. The literature on the variants of meta-heuristic algorithms developed for different problems is quite extensive (Talbi, 2009; Gogna and

ORCID: Rafet DURGUT, https://orcid.org/0000-0002-6891-5851
© Zhejiang University Press 2021

Tayal, 2013).

The original ABC algorithm was first developed to solve single-objective continuous problems, but multi-objective (Akbari et al., 2012), binary (Jia et al., 2014), and combinatorial (Karaboga and Gorkemli, 2011) versions have been developed in due course. The related literature across all honeybees-inspired meta-heuristics was detailed in Rajasekhar et al. (2017). Although the ABC algorithm solves optimization problems via evolution, it has some structural problems. The exploration-exploitation balance is not provided properly with existing operators and mechanisms. In the original ABC algorithm, the exploration phase supported by many mechanisms remains more dominant and the exploitation phase remains more recessive. Only one decision variable is updated at each iteration. This situation differs for the binary versions. The dissimilarity-based binary ABC (DisABC) algorithm, which is a similarity-based binary variant proposed by Kashan et al. (2012), generates a new candidate solution by changing the value of more than one bit. Although the DisABC algorithm provides fast convergence by changing several decision variables, it works slowly in large-scale problems because it contains the mixed-integer linear programming model. Kiran and Gündüz (2013) proposed a new variant of the binary ABC (binABC) algorithm based on XOR operations. A candidate solution is generated by a mechanism derived similar to the original ABC update mechanism. However, Kiran and Gündüz (2013) adjusted the balance of exploration and exploitation with a parameter in the mechanism. However, they did not reveal the effect of this parameter. Santana et al. (2019) proposed a new binary ABC (NBABC) algorithm which uses the flip operator instead of arithmetic operations within the neighborhood. NBABC also adapts the maximum number of dimension changes throughout iterations to improve exploration phases. He et al. (2018) proposed another variant of binary ABC (BABC) for the set-union knapsack problem imposing two modifications: a new full mapping function and probability-based dimension change.

Solving binary problems is very substantial in many fields, in particular computer, mathematics, and economics (Lorena et al., 2008). Therefore, there are arrangements of different meta-heuristic algorithms proposed to solve binary optimization problems in Crawford et al. (2017). Chuang et al. (2008) improved the binary PSO to solve the feature selection problem. Korkmaz and Kiran (2018) made a modification of the artificial algae algorithm that is improved for continuous optimization problems to solve the uncapacitated facility location problem (UFLP).

This paper consists of a modified version of the binABC algorithm (Kiran and Gündüz, 2013) by enhancing the exploitation ability. In this study, we aim to obtain better results in solving binary optimization problems using the ABC algorithm. With this purpose, we propose an improved update mechanism. The proposed mechanism consists of two parts. The first is the determination of the number of decision variables to be changed in each iteration as nonlinear and stochastic, which strengthens the exploration at the early phase and the exploitation in the middle and last phases. The second is the use of an improved update operator to transfer better solutions to the next generations.

2 Artificial bee colony algorithm

The ABC algorithm was developed by Karaboga and Basturk (2007), inspired by the honey bees' food search behavior, and was first applied to continuous optimization problems. The ABC algorithm models the swarm intelligence formed by bees interacting with each other in the bee hive. According to the model, there are three types of bees in the hive: employed, onlooker, and scout bees. These bee types are modeled in a way that each of them is responsible for only one phase. In the first phase, the so-called employed bee phase, each employed bee tries to improve its own food source. In the onlooker bee phase, each onlooker bee works on its own food source in proportion to the quality of its food source. In the scout bee phase, if the onlooker bees fail to provide an improvement in the food source, the scout bees start the search for a new food source.

A food source represents a feasible solution in the model. The fitness function, which depends on the cost function of this solution, expresses nectar information, and is calculated by Eq. (1). Depending on the probability value calculated by Eq. (2), based on this fitness value, the neighborhood operator is applied to the current food source as in Eq. (3). The limit value determined for the implementation of the

scout bee phase is controlled by a variable called a trial, and if this value exceeds the limit value, a new solution is generated by Eq. (4).

$$\mathrm{Fit}(x_i)=\begin{cases}\frac{1}{1+f(x_i)}, & f(x_i)\geq 0,\\ 1+|f(x_i)|, & \text{otherwise},\end{cases} \tag{1}$$

$$p_i=\frac{\mathrm{Fit}(x_i)}{\sum_{j=1}^{N}\mathrm{Fit}(x_j)}, \tag{2}$$

$$v_{i,j}=x_{i,j}+\phi_{i,j}(x_{i,j}-x_{\mathrm{n},j}), \tag{3}$$

$$x_{i,j}=\mathrm{LB}_j+\mathrm{rand}(0,1)(\mathrm{UB}_j-\mathrm{LB}_j), \tag{4}$$

where x_i, x_n, and v_i are the selected, neighbor, and candidate solutions, respectively. $\phi_{i,j}$ is a random number within the range of $[0,1]$. $i=1,2,\cdots,N$ represents the index of the food source, with N as the number of food sources. $j=1,2,\cdots,D$ is the decision variable index in the D-dimensional solution set. LB and UB represent the lower and upper bound values for a decision variable, respectively. There are many studies on detailed analysis of the method and its applications (Karaboga et al., 2014).

3 Binary artificial bee colony

Since the ABC algorithm is developed for the continuous decision variables, it is not possible to apply it to binary optimization problems without making modification. Therefore, the binABC variants have been proposed to solve this problem. Researchers have improved new variants by making some modifications to Eqs. (3) and (4). In what follows, we review three current studies. The methods generate new random solutions using Eq. (5) instead of Eq. (4) by a Bernoulli process:

$$x_{i,j}=\begin{cases}0, & \text{if rand}<0.5,\\ 1, & \text{otherwise}.\end{cases} \tag{5}$$

3.1 binABC

Kiran and Gündüz (2013) modified Eq. (4) with Eq. (6), using the XOR logical operator "$\oplus$" to produce a candidate solution in their binary variant. Here, parameter ϑ is used as a logic NOT gate. If it is less than the threshold value (0.5), then the output of parenthesis is complemented.

$$v_{i,j}=x_{i,j}\oplus\vartheta(x_{i,j}\oplus x_{n,j}), \tag{6}$$

where i is the index of the selected solution, and j is the randomly selected problem dimension. To determine the corresponding bit of the candidate solution, the corresponding bits of the selected and the neighbor solutions are taken to a logical process. Table 1 lists possible candidate bit values that can be obtained using the neighborhood operator. From Table 1, for state 1 ($\vartheta<0.5$), if the input bits are the same, then the output is inverted; otherwise, it follows the input. For state 2 ($\vartheta\geq 0.5$), if the input bits are the same, then the output takes the same value of the input; otherwise, it takes the value of the neighbor bit.

Table 1 Truth table for the XOR-based neighborhood operation

Type	Operation	Bit value			
Input	$x_{i,j}$	0	0	1	1
	$x_{n,j}$	0	1	0	1
	$x_{i,j}\oplus x_{n,j}$	0	1	1	0
	$\vartheta<0.5$ (state 1)	1	0	0	1
	$\vartheta\geq 0.5$ (state 2)	0	1	1	0
Output	$v_{i,j}$ (state 1)	1	0	1	0
	$v_{i,j}$ (state 2)	0	1	0	1

3.2 DisABC

Kashan et al. (2012) proposed a DisABC variant for binary optimization problems. The DisABC method uses a new solution generator by transforming Eq. (4) into the binary search space. The new solution generator calculates dissimilarity between the selected and neighbor solutions using Eqs. (7) and (8). In Eq. (8), the Jaccard similarity coefficient is calculated.

$$\mathrm{sim}(x_i,x_j) = \frac{M_{11}}{M_{01}+M_{10}+M_{11}}, \tag{7}$$

$$\mathrm{dissim}(x_i,x_j) = 1-\mathrm{sim}(x_i,x_j), \tag{8}$$

where M_{11} is the number of bits, both x_i and x_j have value 1. M_{01} and M_{10} are determined similarly according to the bits of x_i and x_j, respectively. The new candidate solution is generated by Eq. (9) and an integer nonlinear programming model (Eq. (10)).

$$\mathrm{dissim}(v_i,x_i)\approx\phi\mathrm{dissim}(x_i,x_j), \tag{9}$$

$$\begin{aligned} \min \quad & |\mathrm{dissim}(v_i, x_i) - \phi \mathrm{dissim}(x_i, x_j)| \\ \text{s.t.} \quad & M_{11} + M_{01} = n_1, \\ & M_{10} \le n_0, \\ & M_{10},\ M_{11},\ M_{01} \ge 0, \\ & M_{10},\ M_{11},\ M_{01} \in \mathbb{Z}, \end{aligned} \tag{10}$$

where ϕ is the random positive scaling factor. The minimum possible value is determined according to the difference between the candidate and selected solutions. Since it is not possible to provide equality in all conditions, there is the approximately equal expression "$\approx$." To solve Eq. (9), it is necessary to solve Eq. (10) first using integer programming techniques. In the constraints, n_1 and n_0 represent the numbers of 1 and 0 bits in the selected solution, respectively. A new candidate solution is generated using the bits of M_{11}, M_{01}, and M_{10} obtained by solving Eq. (10). For more information and examples, the reader can refer to Kashan et al. (2012).

3.3 ABCbin

In this ABC variant proposed by Kiran (2015), continuous decision variables are converted into binary vectors by Eq. (11). In the ABCbin algorithm, Eq. (4) is used as the neighborhood operator. Since it is not possible to apply decision variables in the continuous space to the problem, the binary vector, $\boldsymbol{z}_i$, is used:

$$\boldsymbol{z}_i = \mathrm{round}(|x_i \bmod 2|) \bmod 2. \tag{11}$$

The original ABC algorithm can be easily modified to binary problems by this variant. The ABCbin algorithm was implemented for different population numbers and competitive results against the binABC and DisABC algorithms were obtained (Kiran, 2015).

4 Improved binary artificial bee colony

The ABC algorithm includes updating the value of only one decision variable over D-decision variables in each update process. However, new generation swarm intelligence methods such as WOA and GWO update all decision variables in each iteration. While updating a single decision variable strengthens the exploitation phase, it weakens the exploration phase and raises the problem of getting stuck in a local minimum (Hakli and Kiran, 2020).

Two basic arrangements have been introduced in the proposed binary variant of the ABC algorithm. The first is to increase the convergence rate and strengthen the exploration phase by considering a variable number of bits for the neighborhood operator. Instead of using a fixed value in each iteration, this value is determined by

$$d_t = \mathrm{rand}(0, \alpha) + 0.1D\exp(-t/t_{\max}) + 1, \tag{12}$$

where α is the perturbation coefficient and it is a random integer variable used to prevent exponential decrease. d_t is the number of updated bits. D refers to the problem dimension. t and $t_{\max}$ are the current and maximum iteration numbers, respectively. d_t tends to decrease in each iteration. Therefore, in the first iteration, the exploration phase is strengthened by updating more bits of more selected solutions while a candidate solution is generated. We strengthen the exploitation phase by reducing the number of bits processed when approaching the value of $t_{\max}$. At this stage, we aim to make changes in more than one bit using a perturbation number and to prevent the problem of getting stuck in a local minimum. Since d_t decreases throughout the iteration process, we update more decision variables at the initial stage. Toward the end of the iteration process, fewer decision variables are updated. This modification balances the exploration and exploitation phases.

The proposed method employs the updated version of Eq. (6) as a neighborhood operator. It uses a neighbor or selected solution randomly when we select parameter ϑ in Eq. (6) as 0.5. It does not make any difference if the neighbor solution is better or worse than the selected solution. This approach weakens the exploitation phase because it involves a more random process.

In our proposed variant, parameter ϑ is adaptively determined by

$$\vartheta = \begin{cases} Q_{\max} - \frac{Q_{\max} - Q_{\min}}{t_{\max}} t, & \mathrm{Fit}(x_n) < \mathrm{Fit}(x_i), \\ 0, & \text{otherwise}, \end{cases} \tag{13}$$

where $Q_{\max}$ and $Q_{\min}$ are the upper and lower bounds of a particular range, respectively. If the neighbor solution is better than the current solution, ϑ is assigned zero. The selected bit of the candidate solution is copied from the neighbor solution.

Otherwise, there is a low probability of copying from the neighbor solution. If the neighbor solution is better than the selected solution, state 2 occurs by setting ϑ to be 0. Otherwise, ϑ is determined depending on the iteration process. In the method, ϑ decreases linearly with the iterations. Thus, a worse solution at the beginning of the search is also allowed.

We create IbinABC by adding the two major modifications to binABC. Algorithm 1 presents the pseudocode of IbinABC.

Algorithm 1 IbinABC

Set the parameters (N, $t_{\max}$, $Q_{\max}$, $Q_{\min}$, α, limit)
Generate the initial population
Memorize the best solution
while termination criteria are not met **do**
  for each employed bee **do**
    Select the neighbor using Roulette-Wheel selection
    Determine ϑ using Eq. (13)
    Determine d_t using Eq. (12)
    for $i = 1 : d_t$ **do**
      Apply the neighborhood operator using Eq. (6)
    end for
    Evaluate the candidate solution
    if the candidate solution is better than the selected solution **then**
      Set the current solution as the candidate solution and reset the trial
    else
      Increase the trial
    end if
  end for
  for each onlooker bee according to the probability **do**
    Select the neighbor using Roulette-Wheel selection
    Determine ϑ using Eq. (13)
    Determine d_t using Eq. (12)
    for $i = 1 : d_t$ **do**
      Apply the neighborhood operator using Eq. (6)
    end for
    Evaluate the candidate solution
    if the candidate solution is better than the selected solution **then**
      Set the current solution as the candidate solution and reset the trial
    else
      Increase the trial
    end if
  end for
  if there exists a food source exceeding limit **then**
    Choose one and replace it with the new solution and reset the trial
  end if
  Memorize the best solutions
end while

5 Uncapacitated facility location problem

In this section, we introduce a binary optimization problem named UFLP, which is used to show the effectiveness of the proposed variant. UFLP aims to find the locations of customers whose demands are previously determined and the locations where potential facilities can be built. Each facility has a setup cost and transportation cost between the customer and facility. The main purpose of the problem is to locate the facilities to be built with a minimum total cost and to determine the location of the facilities used by customers. The problem is called "uncapacited" as the facilities are assumed to have the service capacity to meet all the customer demands.

To mathematically define UFLP, let m and n be the number of potential facilities to be built and the number of customers, respectively. Let $\boldsymbol{S}$ be the shipment cost matrix and $S_{i,j}$ be the transportation cost between facility i and customer j. Let $\boldsymbol{F}$ be the setup cost vector. F_i represents the initial installation cost of facility i. $y_{i,j}$ and $x_{i,j}$ are binary decision variables defined as follows:

$$y_{i,j} = \begin{cases} 1, & \text{if customer } j \text{ is served by facility } i, \\ 0, & \text{otherwise,} \end{cases} \tag{14}$$

and

$$x_{i,j} = \begin{cases} 1, & \text{if facility } i \text{ is built,} \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

The objective function is given by

$$\begin{aligned} \min \ & f = \sum_{i=1}^{n}\sum_{j=1}^{m} S_{i,j} y_{i,j} + \sum_{i=1}^{n} F_i x_i \\ \text{s.t.} \ & \sum_{i=1}^{n} y_{i,j} = 1, \quad j = 1, 2, \cdots, m, \\ & y_{i,j} \geq x_i, \quad i = 1, 2, ..., n, \ j = 1, 2, \cdots, m, \\ & y_{i,j}, \ x_i \in \{0, 1\}. \end{aligned} \tag{16}$$

The vector $\boldsymbol{x}$ used as the decision variable in UFLP determines whether the facilities are built or not. If a potential facility is built at its location, then the corresponding decision variable takes the value of 1; otherwise, it takes the value of 0. Since all decision variables of the problem are binary, UFLP belongs to the class of binary integer programming

problems.

OR-Library is a collection of problem instances for a variety of combinatorial optimization problems. Table 2 lists some problem instances including the number of facility locations and the number of customers in OR-Library (Beasley, 1990).

Table 2 OR-Library UFLP dataset description

Problem instance	Problem size	Optimal value
Cap71	16 × 50	932 615.75
Cap72	16 × 50	977 799.40
Cap73	16 × 50	1 010 641.45
Cap74	16 × 50	1 034 976.98
Cap101	25 × 50	796 648.44
Cap102	25 × 50	854 704.20
Cap103	25 × 50	893 782.11
Cap104	25 × 50	928 941.75
Cap131	50 × 50	793 439.56
Cap132	50 × 50	851 495.33
Cap133	50 × 50	893 076.71
Cap134	50 × 50	928 941.75
CapA	100 × 1000	17 156 454.48
CapB	100 × 1000	12 979 071.58
CapC	100 × 1000	11 505 594.33

6 Experimental results

In this section, we discuss the success of the IbinABC algorithm in solving UFLP and give an extensive comparison with the existing methods for the same problem.

6.1 Parameter tuning

We first conducted a parameter tuning process for the experimental study. To determine the best algorithm parameters, we tested several values of $Q_{\max}$, $Q_{\min}$, limit, and the number of individuals in the population (N). Accordingly, we created 24 implementations consisting of three permutations of $Q_{\max}$ and $Q_{\min}$ ($Q_{\max} = 0.5$ and $Q_{\min} = 0.3$, $Q_{\max} = 0.5$ and $Q_{\min} = 0.1$, $Q_{\max} = 0.3$ and $Q_{\min} = 0.1$), four different values of limit ($0.5ND$, ND, $2ND$, and $4ND$), and two values of N (20 and 40), where D is the problem dimension. We determined the parametric values via a preliminary study. We tested these 24 implementations on problem instances CapA, CapB, and CapC for 80 000 function evaluations by running 30 times. Other problem instances have not been taken into the test environment because they are easier to solve, and the results are not distinctive.

Table 3 summarizes the computational results of the 24 implementations tested on the CapA problem instance. The summary of the results over 30 runs consists of the average minimum cost value, the worst minimum cost value, the best minimum cost value, the standard deviation of the cost values, and the number of hits with optimal values. Gap represents the average gap between the optimal value and the obtained mean value for 30 different runs, and it is calculated using

$$\text{Gap} = \frac{\text{Mean} - \text{Optimum}}{\text{Optimum}} \times 100\%, \qquad (17)$$

where "Mean" and "Optimum" denote the average of the best values over 30 different runs and the optimal value, respectively.

From Table 3, the optimal value for 10 implementations was achieved in 30 runs, that is, in each run. We obtained these optimal values 4 times for $N = 20$ and 6 times for $N = 40$. However, we achieved the optimal value 2, 1, 3, and 4 times, respectively, for different limit values in Table 3. The optimal value was reached 2, 2, and 6 times, respectively, for different Q values in Table 3.

Table 4 summarizes the results for the CapB problem instance after 30 runs. According to the table, the best solutions were obtained for $N = 40$, $Q_{\max} = 0.3$, and $Q_{\min} = 0.1$. When the limit parameter was analyzed, the best results were obtained when limit $= ND$ and limit $= 4ND$. Twenty-four hits, i.e., the number of runs with the optimal value, were achieved for $N = 20$, $Q_{\max} = 0.3$, $Q_{\min} = 0.1$, and limit $= 2ND$.

Table 5 summarizes the results for the CapC problem instance after 30 runs. From the table, the optimal value was achieved in 9 out of 30 runs when $N = 40$, limit $= ND$, limit $= 0.5ND$. Thirteen hits were achieved for $N = 20$, $Q_{\max} = 0.3$, $Q_{\min} = 0.1$, and limit $= 2ND$.

From Tables 3–5, the best parameter values were determined as $N = 20$, $Q_{\max} = 0.3$, $Q_{\min} = 0.1$, and limit $= 2ND$. We used the computational results obtained using these parameters for comparison with the other algorithms.

Fig. 1 shows the convergence graphs of the results obtained after 30 runs of different implementations for the CapA problem instance. Each subfigure was obtained for different values of Q, limit, and population. The first four and last four figures

were obtained for $N = 20$ and $N = 40$, respectively. For $N = 20$, if the limit value was not selected properly, there will be a problem of late local convergence and falling into a local minimum (limit $= 0.5ND$ and limit $= ND$). Clearly, $Q_{\max} = 0.3$ and $Q_{\min} = 0.01$ with the other limit values had faster convergence, and the optimal values were achieved. We can observe that there was a decrease in the convergence rate for the other values of Q. Moreover, $Q_{\max} = 0.3$ and $Q_{\min} = 0.01$ had higher convergence speed for $N = 40$ and all values of limit compared to the other values of Q. Fig. 2 shows the convergence graphs of the results obtained after 30 runs of different implementations for the CapC problem instance. The configuration $Q_{\max} = 0.3$ and $Q_{\min} = 0.01$ had higher convergence speed for $N = 40$.

6.2 Comparison of methods

In this subsection, we compared the methods in the literature that are applied to UFLP with a certain success with the proposed method. We obtain the methods and results to be used for comparison directly from Kiran and Gündüz (2013) and Korkmaz

Table 3 Parameter tuning for the CapA problem instance

Limit	Metric	$N = 20$			$N = 40$		
		$Q = \{0.5, 0.3\}$	$Q = \{0.5, 0.1\}$	$Q = \{0.3, 0.1\}$	$Q = \{0.5, 0.3\}$	$Q = \{0.5, 0.1\}$	$Q = \{0.3, 0.1\}$
$0.5ND$	Mean	17 211 513.98	17 211 416.95	17 202 659.73	17 156 454.48	17 157 257.31	17 156 454.48
	Std	118 060.08	121 645.08	124 472.71	0	4397.31	0
	Gap	0.321	0.320	0.269	0	0.005	0
	Hit	23	22	23	30	29	30
ND	Mean	17 158 862.99	17 177 089.92	17 162 797.73	17 157 257.31	17 163 600.57	17 156 454.48
	Std	7349.05	57 841.98	34 743.44	4397.31	34 869.86	0
	Gap	0.014	0.120	0.037	0.005	0.042	0
	Hit	27	25	29	29	28	30
$2ND$	Mean	17 157 257.31	17 156 454.48	17 156 454.48	17 157 257.31	17 157 257.31	17 156 454.48
	Std	4397.31	0	0	4397.31	4397.31	0
	Gap	0.005	0	0	0.005	0.005	0
	Hit	29	30	30	29	29	30
$4ND$	Mean	17 158 862.99	17 156 454.48	17 156 454.48	17 156 454.48	17 162 797.73	17 156 454.48
	Std	7349.05	0	0	0	34 743.44	0
	Gap	0.014	0	0	0	0.037	0
	Hit	27	30	30	30	29	30

Table 4 Parameter tuning for the CapB problem instance

Limit	Metric	$N = 20$			$N = 40$		
		$Q = \{0.5, 0.3\}$	$Q = \{0.5, 0.1\}$	$Q = \{0.3, 0.1\}$	$Q = \{0.5, 0.3\}$	$Q = \{0.5, 0.1\}$	$Q = \{0.3, 0.1\}$
$0.5ND$	Mean	13 040 252.87	13 026 593.82	13 015 214.76	13 000 993.81	12 997 280.3	12 985 106.11
	Std	54 956.31	61 875.43	45 090.86	33 861.62	31 331.36	15 256.78
	Gap	0.471	0.366	0.278	0.169	0.140	0.046
	Hit	9	14	12	18	17	22
ND	Mean	13 017 483	13 014 189.49	12 995 042.72	12 993 104.83	13 000 429.62	12 986 432.62
	Std	47 236.15	43 546.89	38 948.36	32 061.70	33 164.37	20 707.79
	Gap	0.296	0.271	0.123	0.108	0.165	0.057
	Hit	15	15	22	23	18	25
$2ND$	Mean	12 996 226.12	12 992 661.06	12 988 144.53	12 993 190.01	12 992 449.05	12 988 220.02
	Std	31 866.51	28 794.20	23 762.93	26 057.15	28 303.10	20 970.35
	Gap	0.132	0.105	0.070	0.109	0.103	0.070
	Hit	20	20	24	20	21	22
$4ND$	Mean	12 989 864.89	12 997 800.6	12 998 207.46	12 992 871.59	13 001 931.86	12 987 621.8
	Std	24 533.19	33 722.88	32 646.16	27 174.68	34 542.24	23 511.75
	Gap	0.083	0.144	0.147	0.106	0.176	0.066
	Hit	22	17	21	21	16	25

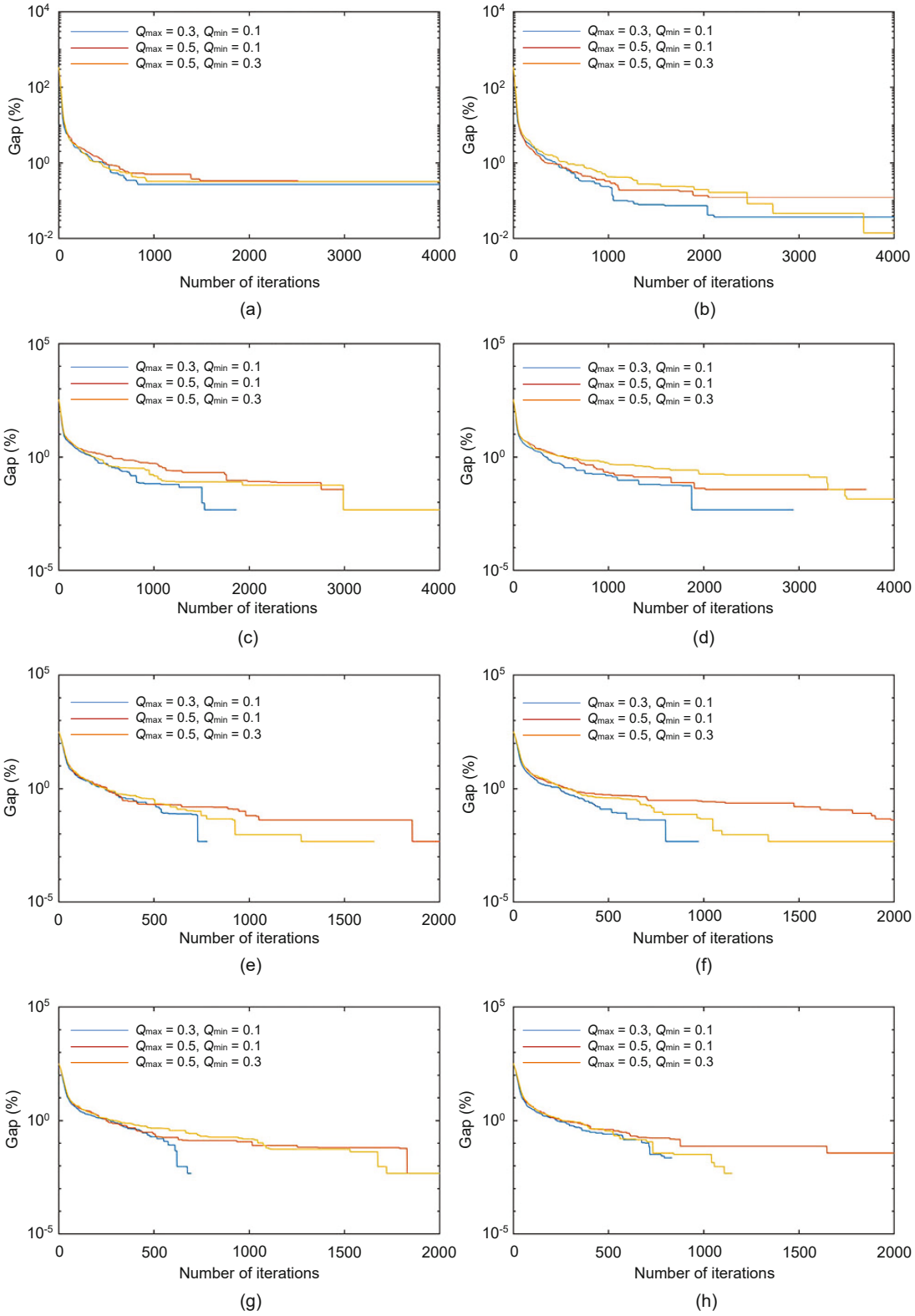

Fig. 1 Parameter tuning on the CapA problem instance: (a) $0.5ND$ for $N = 20$; (b) ND for $N = 20$; (c) $2ND$ for $N = 20$; (d) $4ND$ for $N = 20$; (e) $0.5ND$ for $N = 40$; (f) ND for $N = 40$; (g) $2ND$ for $N = 40$; (h) $4ND$ for $N = 40$ (References to color refer to the online version of this figure)

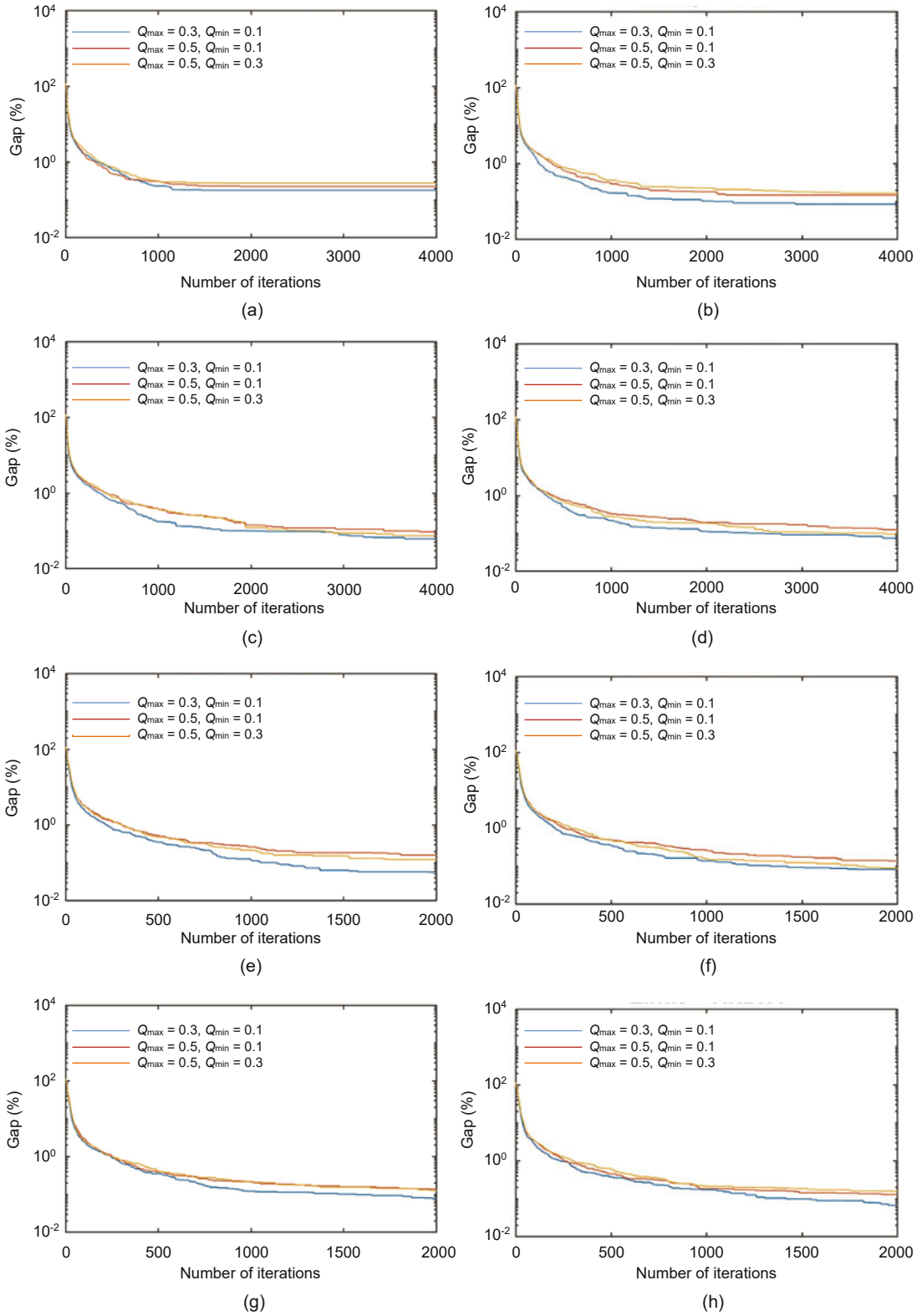

Fig. 2 Parameter tuning on the CapC problem instance: (a) $0.5ND$ for $N = 20$; (b) ND for $N = 20$; (c) $2ND$ for $N = 20$; (d) $4ND$ for $N = 20$; (e) $0.5ND$ for $N = 40$; (f) ND for $N = 40$; (g) $2ND$ for $N = 40$; (h) $4ND$ for $N = 40$ (References to color refer to the online version of this figure)

Table 5 Parameter tuning for the CapC problem instance

Limit	Metric	$N=20$			$N=40$		
		$Q=\{0.5,0.3\}$	$Q=\{0.5,0.1\}$	$Q=\{0.3,0.1\}$	$Q=\{0.5,0.3\}$	$Q=\{0.5,0.1\}$	$Q=\{0.3,0.1\}$
$0.5ND$	Mean	11 537 949.08	11 531 590.6	11 526 389.19	11 519 621.62	11 524 032.61	11 512 198.34
	Std	36 722.90	27 329.10	29 639.45	16 432.87	22 406.63	9798.03
	Gap	0.281	0.226	0.181	0.122	0.160	0.057
	Hit	1	4	7	4	6	9
ND	Mean	11 524 585.71	11 522 732.29	11 515 450.4	11 515 816.97	11 521 586.29	11 515 181.83
	Std	24 926.34	16 770.08	10 456.57	12 328.59	17 861.71	10 868.37
	Gap	0.165	0.149	0.086	0.089	0.139	0.083
	Hit	2	4	6	9	5	4
$2ND$	Mean	11 514 048.28	11 516 527.21	11 512 756.39	11 519 689.91	11 521 590.87	11 514 458.39
	Std	10 001.90	11 814.39	11 326.02	13 314.59	14 937.16	11 717.41
	Gap	0.073	0.095	0.062	0.123	0.139	0.077
	Hit	4	7	13	6	4	7
$4ND$	Mean	11 516 659.14	11 520 120.18	11 514 276.73	11 523 715.66	11 520 523.78	11 513 374.05
	Std	13 096.58	15 931.59	10 349.29	20 569.69	15 529.49	12 098.64
	Gap	0.096	0.126	0.075	0.158	0.130	0.068
	Hit	6	6	6	3	3	8

and Kiran (2018). For a fair comparison between the proposed method and other methods, the size of the population was set to 40 and the maximum number of iterations was set to 2000. Table 6 lists the parameters used in the methods. We implemented the proposed method in the C programming language, and the others were implemented in different platforms. Since the proposed method is faster than the methods in the literature, there will not provide a fair comparison in terms of CPU time (thus we neglect it).

Table 6 Parameter configuration of ABC variants

Method	Population size	Maximum number of iterations	Limit
binABC	40	2000	$0.25ND$
DisABC	40	2000	$2.5ND$
ABCbin	D	1000	$0.5ND$
IbinABC	20	4000	$2ND$

We present the comparison of the IbinABC algorithm with the other ABC algorithm variants in Table 7. According to the table, the proposed method was superior to the other methods for some problem instances. The proposed method obtained the optimal values for all runs except CapB and

Table 7 Comparison with the other binary variants of ABC

Problem instance	binABC				DisABC				ABCbin				IbinABC		
	Gap	Std	R	S	Gap	Std	R	S	Gap	Std	R	S	Gap	Std	R
Cap71	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1
Cap72	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1
Cap73	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1
Cap74	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1
Cap101	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1
Cap102	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1
Cap103	0.00	0.00	1	−	0.00	0.00	1	−	0.01	85.67	4	−	0.00	0.00	1
Cap104	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1
Cap131	0.00	0.00	1	−	0.62	2337.64	4	−	0.20	1065.73	3	−	0.00	0.00	1
Cap132	0.00	0.00	1	−	0.09	813.37	4	+	0.02	213.28	3	+	0.00	0.00	1
Cap133	1215.00	200.24	4	−	0.03	359.03	2	+	0.07	561.34	3	+	0.00	0.00	1
Cap134	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1	−	0.00	0.00	1
CapA	2.96	236 833.50	3	+	0.15	74 782.61	2	−	3.17	268 685.20	4	+	0.00	0.00	1
CapB	2.51	9143.13	2	+	3.30	109 738.50	4	+	2.82	88 452.80	3	+	0.07	23 762.93	1
CapC	2.58	82 312.70	3	+	4.70	95 778.78	4	+	2.04	78 162.20	2	+	0.06	11 326.02	1
Mean			1.53				1.93				2				1

CapC problem instances. However, it had a very low Gap value compared to other methods. The method produced very successful results not only for small-size problem instances but also for large-size ones. Clearly seen in the table, the proposed method was the best among the other methods we compared.

We also compared the IbinABC algorithm with some state-of-the-art meta-heuristic algorithms (GA, binary artificial algae algorithm (binAAA), and binary PSO) and presented the results in Table 8. As can be seen in the table, the results for the CapA, CapB, and CapC problem instances were distinctive. The binAAA and IbinABC algorithms can compete on these three problem instances. The IbinABC algorithm achieved more hits for CapB and CapC than the binAAA algorithm and yielded results closer to the optimal value. We also performed a Wilcoxon signed rank test for all compared methods to show statistical significance at the 95% confidence level. If the p-values are greater than 0.05, it means that the difference between the results of the compared methods is not statistically significant, as indicated by "−" in column S of Tables 7 and 8. Moreover, if the difference between the results of the compared approaches is statistically significant, it is indicated with "+" in column S of Tables 7 and 8. As shown in the tables, the significant difference can be seen in all CapB and CapC problem instances, while no significance is indicated in the other problem case. It is because CapB and CapC are the most difficult problems to solve, which can help make a difference in the performance of algorithms. Looking at ranks of CapB and CapC, it is apparent that IbinABC significantly outperforms all the other approaches. We list the rank values of the methods in column R of Tables 7 and 8.

7 Conclusions

In this paper, we have proposed a binary variant of the ABC algorithm to successfully apply the ABC algorithm to binary optimization problems. The proposed method aims to increase the convergence rate by updating some decision variables in each iteration and to prevent the problem of getting stuck in a local minimum. Another modification is the use of adaptive parameters in XOR-based logical operators. We have conducted a preliminary study to determine these parameters experimentally and obtained the best configuration. We have discussed the success of the proposed binABC algorithm in solving UFLP and presented an extensive comparison with some existing methods. The proposed method showed the best performance in all problem instances of UFLP taken from OR-Library. According to the computational tests, the proposed

Table 8 Comparison with state-of-the-art methods

Problem	GA-SP					BPSO					binAAA					IbinABC			
instance	Gap	Std	Hit	R	S	Gap	Std	Hit	R	S	Gap	Std	Hit	R	S	Gap	Std	Hit	R
Cap71	0.00	0.00	30	1	−	0.000	0.000	30	1	−	0.000	0.000	30	1	−	0.000	0.000	30	1
Cap72	0.000	0.000	30	1	−	0.000	0.000	30	1	−	0.000	0.000	30	1	−	0.000	0.000	30	1
Cap73	0.067	899	19	4	+	0.024	634	26	3	−	0.000	0.000	30	1	−	0.000	0.000	30	1
Cap74	0.000	0.000	30	1	−	0.009	500	29	4	+	0.000	0.000	30	1	−	0.000	0.000	30	1
Cap101	0.068	421	11	4	+	0.043	428	18	3	−	0.000	0.000	30	1	−	0.000	0.000	30	1
Cap102	0.000	0.000	30	1	−	0.010	321	28	4	−	0.000	0.000	30	1	−	0.000	0.000	30	1
Cap103	0.064	505	6	4	+	0.049	521	14	3	+	0.000	0.000	30	1	−	0.000	0.000	30	1
Cap104	0.000	0.000	30	1	−	0.041	1432	28	4	−	0.000	0.000	30	1	−	0.000	0.000	30	1
Cap131	0.068	720	16	3	+	0.171	1505	10	4	+	0.000	0.000	30	1	−	0.000	0.000	30	1
Cap132	0.000	0.000	30	1	−	0.058	1055	21	4	−	0.000	0.000	30	1	−	0.000	0.000	30	1
Cap133	0.091	685	10	4	+	0.083	690	10	3	+	0.000	0.000	30	1	−	0.000	0.000	30	1
Cap134	0.000	0.000	30	1	−	0.195	2594	18	4	+	0.000	0.000	30	1	−	0.000	0.000	30	1
CapA	0.046	22 451	24	3	−	1.691	319 855	8	4	+	0.000	0.000	30	1	−	0.000	0.000	30	1
CapB	0.584	66 658	9	3	+	1.403	135 326	5	4	+	0.248	39 224	15	2	+	0.070	23 762	24	1
CapC	0.705	51 848	2	3	+	1.622	115 156	1	4	+	0.295	29 766	1	2	+	0.062	11 326	13	1
Mean				2.3					3.3					1.1					1.0

method is superior to the other methods.

Compliance with ethics guidelines

Rafet DURGUT declares that he has no conflict of interest.

References

Akbari R, Hedayatzadeh R, Ziarati K, et al., 2012. A multi-objective artificial bee colony algorithm. *Swarm Evol Comput*, 2:39-52. https://doi.org/10.1016/j.swevo.2011.08.001

Askarzadeh A, 2016. A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm. *Comput Struct*, 169:1-12. https://doi.org/10.1016/j.compstruc.2016.03.001

Beasley JE, 1990. OR-Library: distributing test problems by electronic mail. *J Oper Res Soc*, 41(11):1069-1072. https://doi.org/10.1057/jors.1990.166

Chuang LY, Chang HW, Tu CJ, et al., 2008. Improved binary PSO for feature selection using gene expression data. *Comput Biol Chem*, 32(1):29-38. https://doi.org/10.1016/j.compbiolchem.2007.09.005

Crawford B, Soto R, Astorga G, et al., 2017. Putting continuous metaheuristics to work in binary search spaces. *Complexity*, 2017:8404231. https://doi.org/10.1155/2017/8404231

Gogna A, Tayal A, 2013. Metaheuristics: review and application. *J Exp Theor Artif Intell*, 25(4):503-526. https://doi.org/10.1080/0952813X.2013.782347

Hakli H, Kiran MS, 2020. An improved artificial bee colony algorithm for balancing local and global search behaviors in continuous optimization. *Int J Mach Learn Cybern*, 11(9):2051-2076. https://doi.org/10.1007/s13042-020-01094-7

He YC, Xie HR, Wong TL, et al., 2018. A novel binary artificial bee colony algorithm for the set-union knapsack problem. *Fut Gener Comput Syst*, 78:77-86. https://doi.org/10.1016/j.future.2017.05.044

Holland JH, 1992. Genetic algorithms. *Sci Amer*, 267(1):66-73. https://doi.org/10.1038/scientificamerican0792-66

Hussain K, Salleh MNM, Cheng S, et al., 2019. Metaheuristic research: a comprehensive survey. *Artif Intell Rev*, 52(4):2191-2233. https://doi.org/10.1007/s10462-017-9605-z

Jia DL, Duan XT, Khan MK, 2014. Binary artificial bee colony optimization using bitwise operation. *Comput Ind Eng*, 76:360-365. https://doi.org/10.1016/J.CIE.2014.08.016

Karaboga D, Basturk B, 2007. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *J Glob Optim*, 39(3):459-471. https://doi.org/10.1007/s10898-007-9149-x

Karaboga D, Gorkemli B, 2011. A combinatorial artificial bee colony algorithm for traveling salesman problem. Int Symp on Innovations in Intelligent Systems and Applications, p.50-53. https://doi.org/10.1109/INISTA.2011.5946125

Karaboga D, Gorkemli B, Ozturk C, et al., 2014. A comprehensive survey: artificial bee colony (ABC) algorithm and applications. *Artif Intell Rev*, 42(1):21-57. https://doi.org/10.1007/s10462-012-9328-0

Kashan MH, Nahavandi N, Kashan AH, 2012. DisABC: a new artificial bee colony algorithm for binary optimization. *Appl Soft Comput*, 12(1):342-352. https://doi.org/10.1016/J.ASOC.2011.08.038

Kennedy J, Eberhart R, 1995. Particle swarm optimization. Proc Int Conf on Neural Networks, p.1942-1948. https://doi.org/10.1109/ICNN.1995.488968

Kiran MS, 2015. The continuous artificial bee colony algorithm for binary optimization. *Appl Soft Comput*, 33:15-23. https://doi.org/10.1016/J.ASOC.2015.04.007

Kiran MS, Gündüz M, 2013. XOR-based artificial bee colony algorithm for binary optimization. *Turk J Electr Eng Comput Sci*, 21:2307-2328. https://doi.org/10.3906/ELK-1203-104

Korkmaz S, Kiran MS, 2018. An artificial algae algorithm with stigmergic behavior for binary optimization. *Appl Soft Comput*, 64:627-640. https://doi.org/10.1016/J.ASOC.2018.01.001

Lorena AC, de Carvalho ACPLF, Gama JMP, 2008. A review on the combination of binary classifiers in multiclass problems. *Artif Intell Rev*, 30(1-4):19. https://doi.org/10.1007/s10462-009-9114-9

Mallipeddi R, Suganthan PN, Pan QK, et al., 2011. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Appl Soft Comput*, 11(2):1679-1696. https://doi.org/10.1016/J.ASOC.2010.04.024

Mirjalili S, Lewis A, 2016. The whale optimization algorithm. *Adv Eng Softw*, 95:51-67. https://doi.org/10.1016/J.ADVENGSOFT.2016.01.008

Mirjalili S, Mirjalili SM, Lewis A, 2014. Grey wolf optimizer. *Adv Eng Softw*, 69:46-61. https://doi.org/10.1016/J.ADVENGSOFT.2013.12.007

Rajasekhar A, Lynn N, Das S, et al., 2017. Computing with the collective intelligence of honey bees—a survey. *Swarm Evol Comput*, 32:25-48. https://doi.org/10.1016/J.SWEVO.2016.06.001

Rechenberg I, 1978. Evolutionsstrategien. In: Schneider B, Ranft U (Eds.), Simulationsmethoden in der Medizin und Biologie. Medizinische Informatik und Statistik, Vol 8. Springer, Berlin, Heidelberg, p.83-114.

Santana CJJr, Macedo M, Siqueira H, et al., 2019. A novel binary artificial bee colony algorithm. *Fut Gener Comput Syst*, 98:180-196. https://doi.org/10.1016/J.FUTURE.2019.03.032

Storn R, Price K, 1997. Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *J Glob Optim*, 11(4):341-359. https://doi.org/10.1023/A:1008202821328

Talbi EG, 2009. Metaheuristics: from Design to Implementation. John Wiley & Sons, Hoboken, New Jersey, USA.

Wu GH, Mallipeddi R, Suganthan PN, 2019. Ensemble strategies for population-based optimization algorithms—a survey. *Swarm Evol Comput*, 44:695-711. https://doi.org/10.1016/J.SWEVO.2018.08.015