Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2013) 21: 2307 – 2328

doi:10.3906/elk-1203-104

XOR-based artificial bee colony algorithm for binary optimization

Mustafa Servet KIRAN,* Mesut GÜNDÜZ
Department of Computer Engineering, Selçuk University, Alaeddin Keykubad Campus, Konya, Turkey

Received: 23.03.2012 • **Accepted:** 02.08.2012 • **Published Online:** 30.10.2013 • **Printed:** 25.11.2013

Abstract: The artificial bee colony (ABC) algorithm, which was inspired by the foraging and dance behaviors of real honey bee colonies, was first introduced for solving numerical optimization problems. When the solution space of the optimization problem is binary-structured, the basic ABC algorithm should be modified for solving this class of problems. In this study, we propose XOR-based modification for the solution-updating equation of the ABC algorithm in order to solve binary optimization problems. The proposed method, named binary ABC (binABC), is examined on an uncapacitated facility location problem, which is a pure binary optimization problem, and the results obtained by the binABC are compared with results obtained by binary particle swarm optimization (BPSO), the discrete ABC (DisABC) algorithm, and improved BPSO (IBPSO). The experimental results show that binABC is an alternative tool for solving binary optimization problems and is a competitive algorithm when compared with BPSO, DisABC, and IBPSO in terms of solution quality, robustness, and simplicity.

Key words: Swarm intelligence, artificial bee colony, binary optimization, logic operators, uncapacitated facility location

1. Introduction

In recent years, many swarm intelligence methods have been proposed for solving hard optimization problems due to their simple structures and production of effective solutions for problems in a reasonable time. The artificial bee colony (ABC) algorithm, which is one of the most popular swarm intelligence algorithms, was first introduced by Karaboga in 2005 for numerical optimization problems and was inspired by the intelligent behaviors of real honey bee colonies [1]. For solving numerical optimization problems, the ABC was designed using the interaction in the swarm and there are 2 types of bees in the ABC, the employed and unemployed bees. Employed bees try to find food sources that represent a feasible solution for the optimization problem using the interaction in the employed bee population. Employed bees also move the position information about the food sources to the ABC hive. Unemployed foragers consist of onlooker bees and scout bees. The onlooker bees try to improve the solution of the employed bees by considering the information shared by the employed bees. If a food source could not be improved in a certain time, the employed bee of this food source becomes a scout bee. After a new food source is randomly generated for the scout bee, this scout bee becomes an employed bee.

After the invention of the basic ABC algorithm in 2005, many ABC variants that use the model given above have been proposed for solving different optimization problems. Karaboga and Basturk [2] presented the performance analysis of the ABC and compared it with particle swarm optimization (PSO), the genetic algorithm (GA), and the particle swarm-inspired evolutionary algorithm, and they also extended it to constrained

*Correspondence: mskiran@selcuk.edu.tr

optimization problems [3,4]. Karaboga and Basturk [5] used the ABC for solving multidimensional numeric problems and compared its results with PSO, differential evolution, and the evolutionary algorithm. Inspired by PSO, Zhu and Kwong [6] added a component named *gbest* to the solution-updating equation of the ABC algorithm in order to increase the exploitation ability of the ABC. Alatas [7] proposed an ABC model that uses chaotic maps for parameter adaptation so as to improve the convergence characteristics and to prevent the ABC from getting stuck in local minimums. Kashan et al. [8] introduced a binary version of the ABC, named discrete artificial bee colony (DisABC), for pure binary optimization problems; Karaboga and Gorkemli [9] proposed a combinatorial ABC for solving traveling salesman problems; and Kıran et al. analyzed the performance of the ABC with a neighborhood operator on the traveling salesman problem [10] and added a new information-sharing strategy to the basic ABC algorithm [11]. In order to increase the convergence rate of the ABC on the constrained optimization problems and composite and some nonseparable numerical functions, Akay and Karaboga [12] added a parameter called the modification rate to the basic ABC algorithm. The ABC algorithm was also used for test suite optimization [13], training an artificial neural network [14], data clustering [15], dynamic deployment of stationary and mobile sensor networks [16,17], wireless network routing [18], symbolic regression [19], the leaf-constrained minimum spanning tree problem [20], designing of adaptive finite and infinite impulse response filters [21], and reducing the computational complexity of the partial transmit sequence in orthogonal frequency division multiplexing systems [22]. In addition, Karaboga et al. presented a comprehensive survey on the modifications, hybridizations, and applications of the ABC algorithm, and the studies on the ABC conducted from 2005 to 2011 can be found in [23].

According to the literature review, the basic ABC algorithm is a competitive algorithm for optimization problems with continuous solution space. If the solution space of the problem is binary-structured, the basic ABC algorithm must be modified for solving this class of optimization problems. Using the conceptual structure of the basic ABC algorithm and an XOR logic operator, we propose a binary version of the ABC for solving binary optimization problems. The proposed method is examined on a pure binary optimization problem, the uncapacitated facility location problem (UFLP). The UFLP is one of the most widely studied problems in combinatorial optimization. It assumes the minimizing of the total cost of providing the demand of customers under the conditions that are a fixed cost of setting up a facility and a shipping cost of satisfying the customer demand for that facility [24]. The obtained results by the presented method are compared with binary PSO (BPSO) [25], improved BPSO (IBPSO) [26], and the DisABC algorithm [8].

The paper is organized as follows: Section 1 gives the introduction and a brief literature review. The basic ABC algorithm and proposed method are given in Sections 2 and 3, respectively. The BPSO, IBPSO, and DisABC methods are briefly explained in Section 4. Section 5 presents a mathematical model of the UFLP and Section 6 gives the experiments and experimental results. The obtained results are discussed in Section 7, and, finally, the study is concluded in Section 8.

2. The basic ABC algorithm

In the ABC hive, there are 2 types of bees, employed foragers and unemployed foragers [1]. Employed foragers move nectar sources and position information about the food sources to the hive, continuously. There are 2 unemployed foragers in the hive, the onlooker and scout bees. Onlooker bees wait for information shared by the employed foragers in order to search around the food sources of the employed foragers. The occurrence of the scout bee depends on the quality of the food source. If a food source could not be improved in a certain

time (limit parameter of the algorithm determined by the designer), the employed forager of this food source becomes a scout bee. After a new solution produced for the scout bee, this scout bee becomes an employed forager. Moreover, the basic ABC algorithm consists of 4 phases, named initialization, employed, onlooker, and scout bee phases.

2.1. Initialization phase

In this phase, a feasible solution is produced for each employed bee using Eq. (1) and the abandonment counters of the employed bees that will be used for testing the limit are reset.

$$X_i^j = X_j^{min} + R \times \left(X_j^{max} - X_j^{min}\right) \quad i = 1, 2, \ldots, N \, and \, j = 1, 2, \ldots, D \tag{1}$$

Here, X_i^j is the jth dimension of the ith employed bee, and X_j^{min} and X_j^{max} are the lower and upper bounds of the jth dimension, respectively. R is a random number in the range of [0,1], N is the number of employed bees, and D is the dimensionality of the problem.

After producing new solutions for the employed bees, the fitness values of the solutions are calculated using Eq. (2):

$$fit_i = \begin{cases} \frac{1}{1+f_i} & \text{if} \quad (f_i \geq 0) \\ 1 + abs\left(f_i\right) & \text{if} \quad (f_i < 0) \end{cases} , \tag{2}$$

where fit_i is the fitness value of the ith employed bee (or solution) and f_i is the objective function value, specific for the optimization problem, of the ith employed bee.

2.2. Employed bee phase

Each employed bee tries to improve its self-solution using Eq. (3). If the fitness value of the new candidate solution obtained by Eq. (3) is better than the old one, then the employed bee memorizes the new solution and its abandonment counter is reset; otherwise, the abandonment counter of the employed bee is increased by 1 (let $V_i = X_i$):

$$V_i^j = X_i^j + \varphi \times \left(X_i^j - X_k^j\right), \quad ik\epsilon\left\{1, 2, \ldots, N\right\}, \; j\epsilon\left\{1, 2, \ldots, D\right\} \, and \, i \neq k \quad , \tag{3}$$

where V_i^j is the jth dimension of the ith candidate solution, X_i^j is the jth dimension of the ith employed bee, X_k^j is the jth dimension of the neighbor employed bee, and φ is a random number in the range of [–1, + 1]. It should be mentioned that only 1 dimension of the solution of an employed bee is updated at each iteration, and this dimension and neighbor bee chosen from the employed bee population are randomly selected in Eq. (3).

2.3. Onlooker bee phase

The onlooker bees wait for information about food sources that will be shared by the employed bee in the hive. After the employed bees return to the hive, the employed bees share the position information of the self-solutions with onlooker bees in the dance area of the hive. The onlooker bees select an employed bee in order to improve its solution using Eq. (4) [19]:

$$p_i = \frac{0.9 \times fit_i}{fit_{best}} + 0.1, \tag{4}$$

where p_i is the selected probability of the ith employed bee, fit_i is the fitness value of the ith employed bee, fit_{best} is the best solution in the employed bee population, and N is the number of employed bees.

After an employed bee is selected, the solution of the employed bee is updated using Eq. (3). If the new solution is better than the old solution, the solutions are replaced and the abandonment counter of the employed bee is reset. Otherwise, the abandonment counter of the employed bee is increased by 1.

2.4. Scout bee phase

In this phase, the abandonment counter with the maximum content is fixed and this content is compared with the limit. If the content is higher than the limit, the employed bee of this counter becomes a scout bee. If a scout bee occurs at the ABC iteration, a new solution is generated for the scout bee using Eq. (1), the abandonment counter is reset, and the scout bee becomes an employed bee.

Despite the fact that the best solution of the population is not directly used for updating the solution by the employed or onlooker bees for the population, the best solution obtained so far is stored at each iteration. The ABC is an iterative algorithm and the maximum evaluation number or maximum iteration number can be used for termination of the algorithm. In addition, the number of employed bees and onlooker bees is equal to each other and only 1 scout bee can occur at each iteration. After all of the explanations given above, the basic algorithm and framework of the ABC are presented in Figures 1 and 2, respectively.

1. **Initialization Phase**
   - ***For each employed bee***
     - Generate the solution for the employed bee by using Eq.1
     - Calculate the Fitness value of the solutions by using Eq.2
     - Reset the abandonment counters of the employed bee
2. **Employed Bee Phase**
   - ***For each employed bee***
     - Update the solution by using Eq. 3
     - Calculate the fitness value of new solution by using Eq.2.
     - If new solution is better than the old solution, employed bee memorizes new solution and abandonment counter of the employed bee is reset, otherwise the abandonment counter of the employed bee is increased by 1.
3. **Onlooker Bee Phase**
   - ***For each onlooker bee***
     - Calculate the selection probability by using Eq.4
     - Select an employed bee and update its solution by using Eq.3
     - Calculate the fitness value of new solution of onlooker bee.
     - If new solution is better than the old solution, employed bee memorizes new solution and the abandonment counter of the employed bee is reset, otherwise the abandonment counter of the employed bee is increased by 1.
4. **Scout Bee Phase**
   - Fix the abandonment counter with the maximum value.
   - If the content is higher than the limit, generate a new solution for the employed bee by using Eq.1 and reset the abandonment counter of this employed bee.
   - Calculate the fitness value of new solution.
5. **Store the global best solution obtained so far**.
6. **If a termination condition is not satisfied return to Employed Bee Phase.**
7. **Report the best solution obtained.**

Figure 1. The ABC algorithm.

3. The binABC algorithm

When the solution space of the problem is binary-structured, Eqs. (1) and (2) of the basic ABC algorithm should be modified for solving binary optimization problems. In the initialization phase of the binary ABC (binABC), we proposed a Bernoulli process using random numbers for initialization of the employed bee population. For

each dimension of the problem, a random number is generated in the range of [0,1]. If the random number is less than 0.5, the corresponding dimension gets 0; otherwise, it gets 1. This is formulated as follows:

$$P\left(Open\right) = P\left(X_{i,j} = 1\right) = p, \tag{5}$$

$$P\left(Close\right) = P\left(X_{i,j} = 0\right) = 1 - p, \tag{6}$$

$$X_{i,j} = \begin{cases} 0 & \text{if} \quad (r_{i,j} < p) \\ 1 & \text{if} \quad (r_{i,j} \geq p) \end{cases}, \tag{7}$$

where p is the probability value, which is taken as 0.5 for the binABC algorithm, because whether a facility is opened or not causes an equal probability in the initialization phase of the binABC algorithm; $r_{i,j}$ is the trial, which is a randomly generated number in [0,1]; and $X_{i,j}$ is the jth dimension of the ith employed bee.

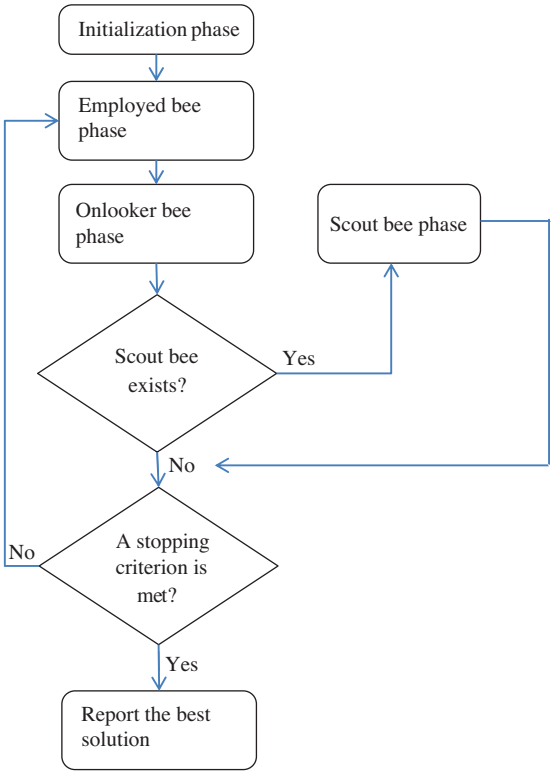

Figure 2. General framework of the ABC.

After the employed bee population is initialized, the positions of the solutions are updated in the employed and onlooker bee phases. The equation of updating the position (Eq. (2)) is modified as follows:

$$V_i^j = X_i^j \oplus \left[\varphi \left(X_i^j \oplus X_k^j\right)\right] \; ik \in \{1, 2, \ldots, N\}, \; j\epsilon\{1, 2, \ldots, D\} \text{ and } i \neq k, \tag{8}$$

where V_i^j is the jth dimension of the ith candidate solution, X_i^j is the jth dimension of the ith employed bee, X_k^j is the jth dimension of the kth employed bee, $\oplus$ is a logic operator, and φ is the logic NOT gate with 50% probability. If φ is less than 0.5, the result obtained by $\left(X_i^j \oplus X_k^j\right)$ is inverted; otherwise, the result is not inverted.

Different logic operators such as OR, AND, or XOR can be used in Eq. (8), but we use XOR as a logic operator because the changing probability of the bit in the solution is 50% in this gate. The reason for using XOR is explained using Tables 1–3.

Table 1. Position update with XOR.

			1	2	XOR with 1	XOR with 2
X_i^j	X_k^j	$\left(X_i^j \oplus X_k^j\right)$	$\varphi < 0.5$	$\varphi \geq 0.5$	$X_i^j \oplus \left[\varphi \left(X_i^j \oplus X_k^j\right)\right]$	$X_i^j \oplus \left[\varphi \left(X_i^j \oplus X_k^j\right)\right]$
0	0	0	1	0	1	0
0	1	1	0	1	0	1
1	0	1	0	1	1	0
1	1	0	1	0	0	1

Table 2. Position update with OR.

			1	2	OR with 1	OR with 2
X_i^j	X_k^j	$\left(X_i^j \oplus X_k^j\right)$	$\varphi < 0.5$	$\varphi \geq 0.5$	$X_i^j \oplus \left[\varphi \left(X_i^j \oplus X_k^j\right)\right]$	$X_i^j \oplus \left[\varphi \left(X_i^j \oplus X_k^j\right)\right]$
0	0	0	1	0	1	0
0	1	1	0	1	0	1
1	0	1	0	1	1	1
1	1	1	0	1	1	1

Table 3. Position update with AND.

			1	2	AND with 1	AND with 2
X_i^j	X_k^j	$\left(X_i^j \oplus X_k^j\right)$	$\varphi < 0.5$	$\varphi \geq 0.5$	$X_i^j \oplus \left[\varphi \left(X_i^j \oplus X_k^j\right)\right]$	$X_i^j \oplus \left[\varphi \left(X_i^j \oplus X_k^j\right)\right]$
0	0	0	1	0	0	0
0	1	0	1	0	0	0
1	0	0	1	0	1	0
1	1	1	0	1	0	1

According to Tables 1–3, if AND or OR is used as the logic operator, the output (V_i^j) is 0 or 1 with a probability of 75%, respectively. In the OR and AND operators, the bits in the candidate solution tend to be 1 and 0, respectively. When we use the XOR as the logic operator in Eq. (8), the changing probability of the bit is 50%. We therefore use the XOR operator as the logic operator in the implementation and experiments. The main difference between the binABC algorithm and the basic ABC algorithm originates from Eq. (8), which is a modified version of Eq. (2), in order to move on the binary-structured solution space.

4. BPSO and DisABC algorithms

4.1. BPSO algorithm

The BPSO algorithm was first proposed by Kennedy and Eberhart in order to solve binary problems [25]. In their method, potential solutions, called particles, fly throughout the solution space to find the optimum solution. Each particle has a velocity and this velocity is updated at each iteration using Eq. (9).

$$v_{ij}\left(t+1\right) = v_{ij}\left(t\right) + r_1 \times c_1 \left(pbest_{ij}(t) - X_{ij}(t)\right) + r_2 \times c_2(gbest_j(t) - X_{ij}(t)) \tag{9}$$

Here, v_{ij} is the velocity of the ith particle on the jth dimension; $pbest_{ij}$ is the best solution obtained by the ith particle in previous iterations; X_{ij} is the jth dimension of the ith particle; $gbest_j$ is the jth dimension of the best solution obtained by the swarm so far; c_1 and c_2 are the positive acceleration constants used to scale the contribution of the cognitive and social components, respectively; r_1 and r_2 are the random numbers in the range of [0,1], which are stochastic elements of the algorithm; $i$is the particle index; j is the dimension index; and $t$is the time step.

After calculating the velocity of each particle, the particle's positions in the BPSO algorithm are updated using Eq. (10):

$$X_{ij}(t+1) = \begin{cases} 1 & \text{if } r_{ij} < sig(v_{ij}(t+1)) \\ 0 & \&otherwise \end{cases}, \tag{10}$$

where r_{ij} is a random number in the range of [0,1] and $sig(v_{ij}(t+1))$ is calculated as follows:

$$sig(v_{ij}(t+1)) = \frac{1}{1+e^{-v_{ij}(t+1)}}. \tag{11}$$

By considering the minimization problems, the personal best solution of the particle and the global best solution of the swarm at the next time step, (t + 1), are calculated using Eqs. (9) and (10), respectively:

$$pbest_i(t+1) = \begin{cases} pbest_i(t), & \text{if } f(X_i(t+1)) \geq f(pbest_i(t)) \\ X_i(t+1), & \text{if } f(X_i(t+1)) < f(pbest_i(t)) \end{cases}, \tag{12}$$

where f is the objective function specific for the minimization problem and $X_i(t+1)$ is the position of the ith particle at time step (t + 1).

$$gbest(t+1) = min\{f(pbest_i(t+1))\} \tag{13}$$

In order to control the exploration and exploitation abilities of the swarm, a new parameter, called the inertia weight (w), was introduced by Shi and Eberhart [27]. The inertia weight controls the momentum of the particle by weighting the contribution of the previous velocity. By adding the inertia weight, Eq. (6) is changed as follows:

$$v_{ij}(t+1) = w \times v_{ij}(t) + r_1 \times c_1\left(pbest_{ij}(t) - X_{ij}(t)\right) + r_2 \times c_2(gbest_j(t) - X_{ij}(t)). \tag{14}$$

4.2. DisABC algorithm

In order to solve pure binary optimization problems, the DisABC algorithm was proposed by Kashan et al. [8]. The DisABC algorithm is based on the conceptual structure of the ABC algorithm. The most important point in the DisABC algorithm is to generate a new solution using the interaction in the bee population. In order to generate a new solution, measuring of the dissimilarity between the employed bee and the neighbor bee is first calculated as follows:

$$Dissimilarity(X_i, X_k) = 1 - Similarity(X_i, X_k) = 1 - \frac{m_{11}}{m_{01}+m_{10}+m_{11}}, \tag{15}$$

where X_i is the ith employed bee and X_k is the neighbor bee selected from the employed bee population. m_{01}, m_{10}, and m_{11} are calculated as follows:

- m_{01} represents the number of bits when $X_{ij} = 0$ and $X_{kj} = 1$, where j = 1,2,...,D and D is the dimensionality of the problem ($m_{01} = \sum_{j=1}^{D} F(X_{ij} = 0,\, X_{kj} = 1)$).
- m_{10} represents the number of bits when $X_{ij} = 1$ and $X_{kj} = 0$, where j = 1,2,...,D and D is the dimensionality of the problem ($m_{10} = \sum_{j=1}^{D} F(X_{ij} = 1,\, X_{kj} = 0)$).
- m_{11} represents the number of bits when $X_{ij} = 1$ and $X_{kj} = 1$, where j = 1,2,...,D and D is the dimensionality of the problem ($m_{11} = \sum_{j=1}^{D} F(X_{ij} = 1,\, X_{kj} = 1)$).

Eq. (2) of the ABC algorithm, to be used to produce a new solution in the continuous solution space, is reshaped as follows:

$$Dissimilarity\,(V_i, X_i) \approx \vartheta \times Dissimilarity\,(X_i, X_k)\,, \tag{16}$$

where V_i is the candidate solution for X_i, ϑ is the positive scaling factor, and $\approx$ is the almost-equal operator. In order to construct V_i, the 3 variables given below must be determined:

- M_{01} represents the number of bits when $V_{ij} = 0$ and $X_{ij} = 1$, where j = 1,2,...,D and D is the dimensionality of the problem ($M_{01} = \sum_{j=1}^{D} F(V_{ij} = 0,\, X_{ij} = 1)$).
- M_{10} represents the number of bits when $V_{ij} = 1$ and $X_{ij} = 0$, where j = 1,2,...,D and D is the dimensionality of the problem ($M_{10} = \sum_{j=1}^{D} F(V_{ij} = 1,\, X_{ij} = 0)$).
- M_{11} represents the number of bits when $V_{ij} = 1$ and $X_{ij} = 1$, where j = 1,2,...,D and D is the dimensionality of the problem ($M_{11} = \sum_{j=1}^{D} F(V_{ij} = 1,\, X_{ij} = 1)$).

$$min \left| \left(1 - \frac{M_{11}}{M_{01} + M_{10} + M_{11}}\right) - \vartheta \times \left(1 - \frac{m_{11}}{m_{01} + m_{10} + m_{11}}\right) \right| \tag{17}$$

$$M_{11} + M_{01} = n_1 \tag{18}$$

$$M_{10} \leq n_0 \tag{19}$$

$$M_{01} M_{10}\,, M_{11} \geq 0 \,and\, integer \tag{20}$$

Here, n_1 and n_0 are the number of 1 and 0 values in the X_i binary vector, respectively. For determining the values of M_{01}, M_{10}, $and\, M_{10}$, the integer mathematical model given by Eqs. (17)–(20) must be solved.

After solving the mathematical model and getting the optimal values of $M_{01} M_{10}$, $and\, M_{10}$, the V_i candidate solution can be obtained using the new binary solution generator (NBSG), which is directly taken from [8] and given in Figure 3.

Step 1. *Compute the value of A through A=* $\boldsymbol{\vartheta \times Dissimilarity(X_i, X_k)}$ *and use it in the mathematical programming model (17)-(20) with output* $\boldsymbol{M_{01}, M_{10}}$ ***and*** $\boldsymbol{M_{11}}$*. Apply the total enumeration (TE) scheme to solve the mathematical programming problem optimmaly. Initialize by a* $\boldsymbol{1 \times D}$ *zero solution vector.*
Step 2-1 *(Inheritance phase). Based on any logic, select* $\boldsymbol{M_{11}}$ *number of zero bits from* $\boldsymbol{V_i}$ *which their corresponding value in* $\boldsymbol{X_i}$ *is 1. Change the value of the selected bits from 0 to 1.*
Step 2-2 *(Disinheritance phase). Based on any logic, select* $\boldsymbol{M_{10}}$ *number of zero bits from* $\boldsymbol{V_i}$ *which their corresponding value in* $\boldsymbol{X_i}$ *is 0. Change the value of the selected bits from 0 to 1. Then, report the new binary solution vector* $\boldsymbol{V_i}$ *as output.*

Figure 3. The NBSG algorithm [8].

The workings of the NBSG algorithm, which is the most important part of the DisABC algorithm, is explained with the example below.

Let X_i= {1110010101, X_k= {1010101101 and $\vartheta = 0.6$

$$m_{01}= 2$$

$$m_{10}= 2$$

$$m_{11} = 4$$

$$Dissimilarity\,(X_i, X_k) = 1 - \frac{m_{11}}{m_{01} + m_{10} + m_{11}} = 1 - \frac{4}{2+2+4} = 0.5$$

$$A = \vartheta \times Dissimilarity\,(X_i, X_k) = 0.6 \times 0.5 = 0.3$$

According to the integer mathematical model given by Eqs. (17)–(20):

$$\min f = \left| \left(1 - \frac{M_{11}}{M_{01} + M_{10} + M_{11}}\right) - 0.3 \right|,$$

$$M_{11} + M_{01} = 6,$$

$$M_{10} \leq 4,$$

$$M_{01} M_{10}\,, M_{11} \geq 0\, and\, integer.$$

After the model is solved using the total enumeration scheme, the optimal output is obtained as $M_{01} = 1, M_{10} = 1$, and $M_{11} = 5$ and $f = 0.0143$. After all, V_i is generated as a 1×10 zero-bit array.

Using the NBSG algorithm;

Inheritance: The positions of the bits that are equal to 1 in X_i are found ($Ones_{X_i} = \{1, 2, 3, 6, 8, 10\}$) and $M_{11} = 5$ bit positions are selected from the $Ones_{X_i}$ (assume that the random selection includes {1,3,6,8,10}), and the bits of V_i in these positions are changed to 1. The new state of $V_i\, is\, no\, longer$ {1010010101.

Disinheritance: The positions of the bits that are equal to 0 in X_i are found ($Zeros_{X_i} = \{4, 5, 7, 9\}$) and $M_{10} = 1$ bit position is selected from the $Zeros_{X_i}$ (assume that the random selection includes {4}), and the bit of V_i in these positions is changed to 1. After this change, the final state of the candidate solution $V_i\, is$ {1011010101.

More explanations for the DisABC can be seen in [8]. Despite the fact that the DisABC was used by being hybridized with a local search module for solving the binary optimization problem (UFLP) in [8], we implement the DisABC without a local search module in order to make a clear comparison and show the performance and accuracy of the methods.

4.3. IBPSO

Yuan et al. proposed the IBPSO algorithm for solving unit commitment problems [26]. The particles in the IBPSO start to search the solution space with binary values and the velocities of these particles are also binary values. The main difference between BPSO and IBPSO is that the velocities are not updated on the continuous solution space and the velocities of the particles in IBPSO are the unchanging probabilities of a value in the

particles. In IBPSO, the velocities of the particles are updated using Eq. (21), which is a modified version of the basic velocity equation of BPSO.

$$V_{i,j}(t+1) = \left[\omega_1 \otimes \left(pbest_{i,j}(t) \oplus X_{i,j}(t)\right)\right] + \left[\omega_2 \otimes \left(gbest_j(t) \oplus X_{i,j}(t)\right)\right]$$

$$i = 1,..,N \, and \, j = 1,\ldots D \tag{21}$$

Here, $V_{i,j}(t+1)$ is the jth dimension of the velocity of the ith the particle at time step t + 1; $pbest_{i,j}(t)$ is the jth dimension of the personal best solution of the ith particle at time step t; $gbest_j(t)$ is the jth dimension of the best solution of the swarm at time step t; $X_{i,j}(t)$ is the jth dimension of the ith particle at time step t; ω_1 and ω_2 are binary integer random numbers in the range of [0,1]; N is the number of particles; D is the dimensionality of the problem; and the $\otimes$, $\oplus$, and + operators denote the AND, XOR, and OR logic operators, respectively. After a velocity is obtained from Eq. (21), the particle position is updated as follows:

$$X_{i,j}(t+1) = X_{i,j}(t) \oplus V_{i,j}(t+1). \tag{22}$$

Briefly, while the basic PSO algorithm works on the continuous solution space and the velocities of the BPSO algorithm are on the continuous solution space, the IBPSO algorithm works completely on the binary-structured solution space.

5. Mathematical model of UFLP

A brief description of the UFLP is given in this section because the performance and accuracy of the proposed binABC algorithm are tested on UFLPs. In the basic formulation, the UFLP consists of a set of potential facility sites I, where a facility can be opened, and a set of customer locations J that must be serviced. The goal is to find a subset F of the I facilities that are the corresponding demand of customers J. The objective function of this problem is to minimize the sum of the shipment costs between F and J and the opening costs of the facilities. The general model of the UFLP can be mathematically expressed as [28]:

$$f(UFLP) = min \oplus \left\{ \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{i \in I} f_i y_i \right\}, \tag{23}$$

subject to:

$$\sum_{i \in I} x_{ij} = 1, \; j \in J, \tag{24}$$

$$x_{ij} \leq y_i, \; i \in I \, and \, j \in J, \tag{25}$$

$$x_{ij} \in \{0,1\}, \; i \in I \, and \, j \in J, \tag{26}$$

$$y_i \in \{0,1\}, \; i \in I, \tag{27}$$

where I = {1,2,...,n} is the set of possible facility locations, J = {1,2,...,m} is the set of customer demand points, f_i is the fixed cost of opening a facility in $i \in I$, and c_{ij} is the shipment cost between the ith facility location and the jth customer point. The decision variable x_{ij} is the demand of customer j corresponding to the ith facility, and y_i is the binary variable: $y_i = 1$ if a facility is located in $i \in I$; otherwise, $y_i = 0$.

The constraint in Eq. (24) aims to satisfy all of the demands of the customers. The constraint in Eq. (25) ensures that a customer can be served from a facility only if a facility is opened. Constraints of Eqs. (26) and (27) define the decision variables in the binary structure. Due to the single assignment property of the UFLP, the demand of a customer is always entirely fulfilled by its nearest facility [29]. When the locations of the facilities that will be opened are determined, knowledge of which customer will be served by a facility can be obtained easily. Therefore, we address a vector A of n variables, where each variable in A is 1 bit. Each bit (A_i, i = 1,2,...,n) indicates whether a facility is opened at location i or not. This vector plays a critical role in the binABC because each employed bee represents this vector.

The UFLP is one of the most important NP-hard problems in location theory [8,30,31] and many studies have been presented on it in the literature. In order to solve UFLPs, many exact methods, such as branch-and-bound [32], linear programming and Lagrangian relaxation [33], and dual approach [34], have been proposed. Despite the fact that these methods ensure optimality, the computation time of these methods may be too much. For this reason, some approximate methods have been proposed for solving UFLPs. These methods cannot guarantee the finding of the optimal solution, but they can obtain optimum or near-optimum solutions in a reasonable amount of time. For instance, the GA [35], tabu search [36,37], and discrete and continuous PSO algorithms [24,38] have been proposed in order to solve UFLPs. In addition, a comprehensive study on the UFLP can be found in [39].

We use the UFLP to test the performance and accuracy of the binABC algorithm because: 1) the UFLP is a pure binary optimization problem and there is no continuous or integer variable in the problem, 2) the method does not need to eliminate the infeasibility in the solution produced for the problem, 3) the optimal solutions for the test suit are available, and 4) one of the methods that is used for comparison (DisABC) is also tested on the UFLPs. The binABC and UFLP are described above, and the flowchart of the binABC adapted for solving the UFLP is given in Figure 4.

6. Experiments

In order to test the performance and accuracy of our binABC algorithm, we used the uncapacitated facility location test suit (15 test problems) taken by the OR-Library [40]. In the test suite, 4 problems (Cap71–74) are small-sized, 8 problems (Cap101–104 and Cap131–134) are medium-sized, and the rest of problems (CapA, CapB, and CapC) are large-sized, and the sizes and the costs of the optimal solutions for the problems are given in Table 4.

6.1. Analysis of the control parameters of the binABC

Before the comparison of the binABC with BPSO, the DisABC, and IBPSO, we analyze the effect of the control parameters of the binABC on the Cap71–74, Cap101–104, and Cap131–134 UFLPs. There are only 2 control parameters, the population size and the limit in the binABC algorithm. The limit value proposed for the continuous ABC [41] is expressed as:

$$limit = D \times N, \tag{28}$$

where D is the dimensionality of the problem (number of potential facility locations for the UFLP) and N is the number of employed bees. Because the problem structure and the method are different, we calculate 4 limit values for each population size.

$$lb4 = D \times N/4 \tag{29}$$

Initialization Phase
Determine population size.
Assign half of the population as employed bee, and other half the population as onlooker bee
Determine limit value for this population
Determine the stopping criterion
Load UFLP.
For each employed bee
Generate a random solution which consists of binary value for the employed bee using Eq. 7
Calculate cost of solution of employed bee (Eq. 23)
Calculate fitness value of the solution based on cost value (Eq. 2)
Reset abandonment counter of the employed bee
REPEAT
Employed Bee Phase
For each employed bee
Update a facility status in solution of employed bee (Eq. 8)
Calculate cost of new solution (Eq. 23)
Calculate fitness value of the new solution based on cost value (Eq. 2)
If cost of new solution is lower than old solution, replace them, otherwise increase the abandonment counter by 1
Onlooker Bee Phase
Select an employed bee by using fitness value. (Eq. 4)
Update a facility status in solution of employed bee. (Eq. 8)
Calculate cost of new solution (Eq. 23)
Calculate fitness value of the new solution based on cost value (Eq. 2)
If cost of new solution is lower than old solution, replace them, otherwise increase the abandonment counter by 1
Scout Bee Phase
If the abandonment counter with maximum content is higher than limit value, assign the employed bee with maximum content as scout bee
Generate a random solution which consists of binary value for scout bee using Eq.7
Calculate cost value of the generated solution. (Eq. 23)
Calculate fitness value of generated solution based on cost value (Eq. 2)
Reset abandonment counter of this bee.
Assign this scout bee as employed bee.
Store the global best solution of the population.
UNTIL(The stopping criterion is met)
Report the best solution obtained by the bees.

Figure 4. The detailed binABC algorithm adapted for solving the UFLP.

$$lb2 = D \times N/2 \tag{30}$$

$$lb = D \times N \tag{31}$$

$$lc2 = D \times N \times 2 \tag{32}$$

$$lc4 = D \times N \times 4 \tag{33}$$

The other control parameter of the method is the population size. In the analyses of the control parameters, the population size is taken as 10, 20, 30, 40, and 50. For all of the population sizes and limit values, the maximum iteration number used for the termination condition of the algorithm is taken as 2000 and the binABC algorithm is run 30 times with random seeds. The GAP, which is the difference between the cost of the optimal solution and the solution found by the method, and the standard deviations obtained by the runs are reported in Tables 5–9. For the mean of cost values obtained by 30 runs, the GAP is calculated as follows:

Table 4. Description of the test suite.

Problem name	Problem size	Cost of the optimal solution
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

$$GAP\,(\%) = \frac{f(mean) - f(opt)}{f(opt)} \times 100, \tag{34}$$

where $f(opt)$ is the cost of the optimal solution and $f(mean)$ is the mean of the costs obtained by 30 runs.

As seen from Tables 5–9, when the population size for the algorithm is taken as 40, the best results for all of the problems are obtained. When the limit value for the population is increased, higher quality solutions are not obtained because the search ability of the population decreases. Moreover, when the population size is taken as 50 and the limit value is on the level of *lc4*, the diversity in the population is not kept and the selection of a good neighbor employed bee is made difficult. Therefore, higher quality solutions are not obtained under high limit value and population size conditions. According to the average GAP values and standard deviations, it is seen that the average GAP values in Tables 5 and 6 are the same when the population size is taken as 40, but the results in Table 5 are more robust than the results in Table 6. We then use these (Pop_Size = 40 and *lb4*) conditions for the algorithm for the comparison of the binABC algorithm with the other methods.

Table 5. Results obtained under the *lb4* limit value and different population sizes.

Problem	Pop_Size = 10		Pop_Size = 20		Pop_Size = 30		Pop_Size = 40		Pop_Size = 50	
	Std. dev.	GAP	Std. dev.	GAP	Std. dev.	GAP	Std. dev.	GAP	Std. dev.	GAP
Cap71	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap72	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap73	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap74	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap101	0.00	0.00	428.66	0.04	0.00	0.00	0.00	0.00	0.00	0.11
Cap102	0.00	0.15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap103	435.75	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap104	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap131	405.56	0.29	861.40	0.10	219.63	0.24	0.00	0.00	261.98	0.25
Cap132	439.16	0.06	421.27	0.04	119.82	0.02	0.00	0.00	166.50	0.03
Cap133	511.01	0.09	372.28	0.14	298.61	0.04	0.00	0.12	311.18	0.04
Cap134	97.83	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Avg. (GAP)		0.05		0.03		0.03		0.01		0.04

Table 6. Results obtained under the *lb2* limit value and different population sizes.

Problem	Pop_Size = 10		Pop_Size = 20		Pop_Size = 30		Pop_Size = 40		Pop_Size = 50	
	Std. dev.	GAP	Std. dev.	GAP	Std. dev.	GAP	Std. dev.	GAP	Std. dev.	GAP
Cap71	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap72	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap73	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap74	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap101	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap102	0.00	0.15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap103	415.27	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap104	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap131	383.02	0.30	855.17	0.11	323.74	0.26	397.92	0.02	337.28	0.26
Cap132	436.75	0.05	632.39	0.06	291.02	0.04	133.53	0.01	327.21	0.04
Cap133	422.17	0.08	505.42	0.13	345.44	0.06	273.86	0.13	348.64	0.06
Cap134	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Avg. (GAP)		0.05		0.02		0.03		0.01		0.03

Table 7. Results obtained under the *lb* limit value and different population sizes.

Problem	Pop_Size = 10		Pop_Size = 20		Pop_Size = 30		Pop_Size = 40		Pop_Size = 50	
	Std. dev.	GAP	Std. dev.	GAP	Std. dev.	GAP	Std. dev.	GAP	Std. dev.	GAP
Cap71	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap72	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap73	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap74	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap101	0.00	0.00	350.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00
Cap102	0.00	0.15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap103	453.28	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap104	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap131	396.65	0.30	1030.47	0.13	392.69	0.28	727.24	0.06	261.98	0.25
Cap132	475.12	0.06	479.43	0.05	620.86	0.05	272.23	0.02	200.24	0.03
Cap133	444.47	0.07	584.87	0.15	477.98	0.07	252.61	0.13	503.58	0.07
Cap134	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Avg. (GAP)		0.05		0.03		0.03		0.02		0.03

Table 8. Results obtained under the *lc2* limit value and different population sizes.

Problem	Pop_Size = 10		Pop_Size = 20		Pop_Size = 30		Pop_Size = 40		Pop_Size = 50	
	Std. dev.	GAP	Std. dev.	GAP	Std. dev.	GAP	Std. dev.	GAP	Std. dev.	GAP
Cap71	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap72	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap73	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap74	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap101	0.00	0.00	428.66	0.04	0.00	0.00	0.00	0.00	0.00	0.11
Cap102	0.00	0.15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap103	496.70	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap104	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap131	527.26	0.32	1517.80	0.27	357.95	0.26	1075.24	0.10	360.10	0.26
Cap132	684.52	0.09	658.90	0.06	419.29	0.06	423.63	0.03	414.31	0.05
Cap133	433.39	0.09	1300.91	0.19	480.09	0.09	368.93	0.14	387.44	0.09
Cap134	163.49	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Avg. (GAP)		0.06		0.05		0.03		0.02		0.04

Table 9. Results obtained under the *lc4* limit value and different population sizes.

Problem	Pop_size = 10		Pop_size = 20		Pop_size = 30		Pop_size = 40		Pop_size = 50	
	Std. dev.	GAP	Std. dev.	GAP	Std. dev.	GAP	Std. dev.	GAP	Std. dev.	GAP
Cap71	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap72	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap73	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap74	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap101	0.00	0.00	386.94	0.03	218.26	0.01	0.00	0.00	0.00	0.11
Cap102	0.00	0.15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap103	466.38	0.07	0.00	0.00	105.35	0.01	91.96	0.01	78.15	0.00
Cap104	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cap131	817.66	0.38	1321.66	0.22	516.17	0.30	860.40	0.07	371.85	0.27
Cap132	1793.61	0.19	946.09	0.10	757.16	0.07	464.00	0.05	549.11	4.98
Cap133	1609.71	0.19	1505.94	0.26	810.38	0.13	342.29	0.15	549.11	0.09
Cap134	494.87	0.02	0.00	0.00	135.94	0.00	0.00	0.00	945.59	0.02
Avg. (GAP)		0.08		0.05		0.04		0.02		0.46

The convergence graphs for the algorithm are designed as Cap132 problems under 2 sets of conditions. In the first, the population size is taken as 10, 20, 30, 40, and 50 and the limit value is *lb4* (Figure 5); in the second, the population size is taken as 40 and the limit value is *lb4*, *lb2*, *lb*, *lc2*, and *lb4* (Figure 6).

As seen from Figures 5 and 6, the convergence of the binABC algorithm to optimum or near-optimum solutions is better under the lower limit value condition and the limit value of the binABC is more effective than the population size for finding the optimal or near-optimal solutions.

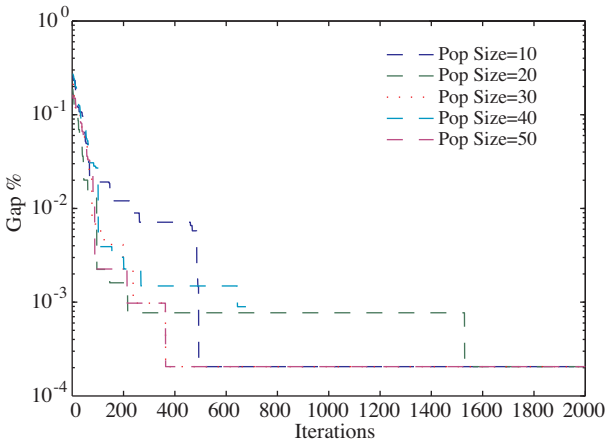

Figure 5. Convergence graph of binABC with *lb4* limit under the different population sizes.

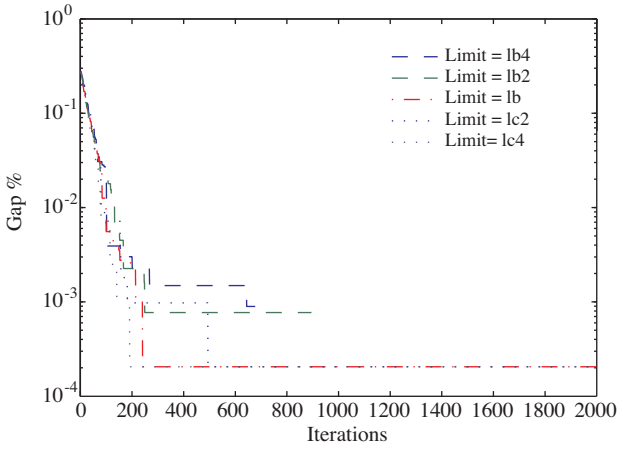

Figure 6. Convergence graph of binABC with population size = 40 under the different limit values.

6.2. Comparing the binABC with BPSO, IBPSO, and the DisABC

In order to make a clear and consistent comparison, the common control parameters of the methods (population size and maximum iteration number) are selected as 40 and 2000, respectively. The algorithms are implemented with MATLAB 2011a and run on an IBM-compatible PC with an i5 2.0-GHz microprocessor and 4 GB of RAM. For each problem, the methods are repeated 30 times with random seeds and the results obtained by the methods are reported as the worst, mean, best, and GAP of the mean. The peculiar control parameters of the methods are set as explained below.

Parameter setting of the BPSO: The upper bound and lower bound of the velocities of the particles is taken as 6 and –6, respectively. The sigmoid function is given in Eq. (11) and values are demonstrated in Figure 7 for velocities between –6 and 6. As seen from Figure 7, the changing probability of a value in the particle vector obtained by Eq. (11) is between 0 and 1. The –6 and 6 values used for the lower and upper bound of the velocities are sufficient practically. In addition, these bounds were used in the first study on BPSO [25]. The inertia weight [27] for the BPSO is calculated as follows:

$$w(t) = \frac{maxit - t}{maxit}, \tag{35}$$

where w is the inertia weight, $maxit$ is the maximum iteration number (2000), and t is the iteration index.

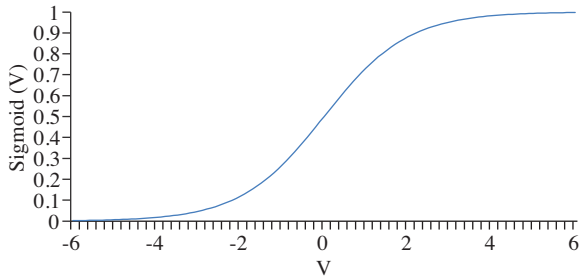

Figure 7. Change probability of a value in the particle vector under the velocity bounds in [–6,6].

Parameter setting of the DisABC: The peculiar control parameters of the DisABC were analyzed in [8] and the selection of the parameters is done according to their work. The scaling factor of the method (ϑ) is calculated using Eq. (36):

$$\vartheta(t) = \vartheta_{\max} - \left(\frac{\vartheta_{\max} - \vartheta_{min}}{maxit}\right) t, \tag{36}$$

where $\vartheta(t)$ is the scaling factor at time step t, $maxit$ is the maximum iteration number, and $\vartheta_{\max}$ and ϑ_{min} are the upper and lower bounds of the scaling factor (0.9 and 0.5), respectively. The *limit* parameter for the population of the DisABC is calculated as follows [8]:

$$limit = 2.5 \times Nb \times D, \tag{37}$$

where Nb is the number of employed bees and D is the number of potential facility locations. It should be mentioned that the exploration and exploitation abilities of the basic ABC algorithm have been balanced using the limit value, but the effects of this parameter on the DisABC algorithm have not yet been analyzed. Therefore, we obtain the limit value using Eq. (37) for the DisABC algorithm.

In [8], random and greedy selection logics were proposed in the inheritance and disinheritance steps of the NBSG algorithm and the effects of these selection logics on the DisABC were analyzed. Due to the fact that results of the random selection logic are better than those of the greedy selection logic, the random selection logic is used in the implementation. Because the performance and accuracy of the methods are investigated, instead of hybridization of the DisABC with the local search module, only the DisABC is used and the parameters of the local search module are not given in this study.

Parameter setting of the binABC: The binABC algorithm has only one peculiar control parameter (*limit*), and it is analyzed in Section 6.1. According to our analysis, the limit value for the binABC is obtained

using Eq. (38):

$$limit = \frac{N \times D}{4}, \tag{38}$$

where N is the population size and D is the potential facility locations.

The accuracy and robustness of the methods are compared according to the GAP values and standard deviations, respectively. In the comparison tables (Tables 10–12), the method with the lowest GAP value and standard deviation is given in bold font.

As seen from Table 10, the BPSO and binABC have an equal performance for the Cap71 and Cap72 problems. BPSO is better than the binABC for the Cap133, CapA, CapB, and CapC problems, and the binABC is better than BPSO for the rest of the problems. Based on the standard deviations of the methods, the binABC is more robust than BPSO for all of the problems. As seen from Table 11, the DisABC algorithm is better than the binABC for the Cap133 and CapA problems. The binABC algorithm is better than the DisABC for the Cap131, Cap132, CapB, and CapC problems, and for the rest of the problems, the performance of the methods are equal to each other. Based on the standard deviations, the robustness of the binABC algorithm is better than that of the DisABC for all of the test problems, except for CapA. Finally, when the binABC algorithm is compared with IBPSO, the binABC algorithm is better than IBPSO for both solution quality and robustness, according to Table 12. The CR column in Tables 10–12 shows the changing rate of the mean results obtained by the binABC with respect to the other methods. If the mean result obtained by binABC is equal to the mean result obtained by the other method, CR is 0. If CR is a negative number, the binABC is worse than the other method. If CR is a positive number, the binABC is better than the other method. CR is calculated as follows:

$$CR\ (\%) = \frac{f_{other}\,(mean) - f_{binabc}(mean)}{f_{other}\,(mean)}, \tag{39}$$

where $f_{other}\,(mean)$ is the mean cost value obtained by one of the other methods (BPSO, DisABC, or IBPSO) and $f_{binabc}(mean)$ is the mean cost value obtained by the binABC algorithm.

A run-time comparison of the methods is also performed and shown in Figure 8, where it can be seen that the running times of the methods depend on the dimension of the problem. The IBPSO algorithm has a lower running time than the BPSO, DisABC, and binABC methods for the small and medium problems. The BPSO and binABC have an equal performance for small- and medium-sized problems. The binABC has a lower running time than the other methods for large-sized problems but the BPSO algorithm has a lower running time than the binABC, DisABC, and IBPSO for the very large-sized problems.

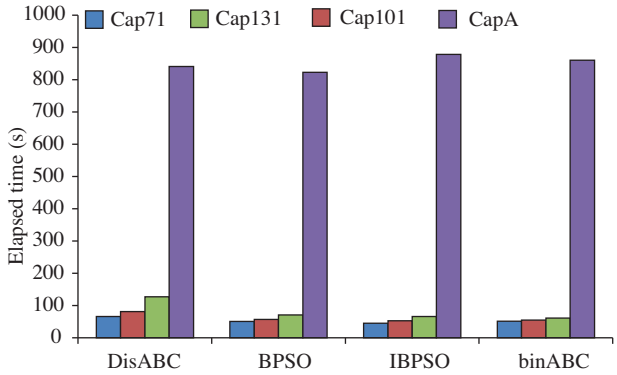

Figure 8. Running time comparison of the methods.

Table 10. Comparison of BPSO and binABC in terms of solution quality and robustness.

Problem	BPSO					binABC					
	Best	Worst	Mean	GAP	Std. dev.	Best	Worst	Mean	GAP	Std. dev.	CR
Cap71	932,615.75	932,615.75	932,615.75	**0.0000**	**0.00**	932,615.75	932,615.75	932,615.75	**0.0000**	**0.00**	0.00
Cap72	977,799.4	977,799.4	977,799.4	**0.0000**	**0.00**	977,799.4	977,799.4	977,799.4	**0.0000**	**0.00**	0.00
Cap73	1,010,886.187	1,012,476.975	1,010,886.187	0.0242	634.62	1,010,641.45	1,010,641.45	1,010,641.45	**0.0000**	**0.00**	0.02
Cap74	1,035,068.312	1,037,717.075	1,035,068.312	0.0088	500.27	1,034,976.975	1,034,976.975	1,034,976.975	**0.0000**	**0.00**	0.01
Cap101	797,016.6558	799,092.1125	797,016.6558	0.0462	566.44	796,648.4375	796,648.4375	796,648.4375	**0.0000**	**0.00**	0.05
Cap102	854,830.955	855,971.75	854,830.955	0.0148	386.76	854,704.2	854,704.2	854,704.2	**0.0000**	**0.00**	0.01
Cap103	894,159.4667	895,027.1875	894,159.4667	0.0422	485.26	893,782.1125	893,782.1125	893,782.1125	**0.0000**	**0.00**	0.04
Cap104	929,694.4467	934,586.975	929,694.4467	0.0810	1951.81	928,941.75	928,941.75	928,941.75	**0.0000**	**0.00**	0.08
Cap131	794,484.4496	797,735.5375	794,484.4496	0.1317	1207.63	793,439.5625	793,439.5625	793,439.5625	**0.0000**	**0.00**	0.13
Cap132	852,273.2071	855,328.675	852,273.2071	0.0914	1196.19	851,495.325	851,495.325	851,495.325	**0.0000**	**0.00**	0.09
Cap133	894,072.9054	896,661.5625	894,072.9054	**0.1115**	821.28	894,095.7625	894,752.025	894,161.3888	0.1215	**200.24**	-0.01
Cap134	930,192.0425	934,586.975	930,192.0425	0.1346	2285.42	928,941.75	928,941.75	928,941.75	**0.0000**	**0.00**	0.13
CapA	17,530,210.57	18,682,895.54	17,530,210.57	**2.1785**	374,302.81	17,180,539.56	18,030,263.31	17,664,663.43	2.9622	**236,833.5**	-0.77
CapB	13,232,039.15	13,633,079.9	13,232,039.15	**1.9490**	176,206.07	13,100,041.02	13,476,652.7	13,304,594.27	2.5081	**91,430.13**	-0.55
CapC	11,676,684.07	11,871,643.26	11,676,684.07	**1.4870**	92,977.85	11,535,255.5102	11,867,887.0012	11,802,532.8641	2.5800	**82,312.70**	-1.08

Table 11. Comparison of DisABC and binABC in terms of solution quality and robustness.

Problem	DisABC					binABC					
	Best	Worst	Mean	GAP	Std. dev.	Best	Worst	Mean	GAP	Std. dev.	CR
Cap71	932,615.75	932,615.7500	932,615.75	**0.0000**	**0.00**	932,615.75	932,615.75	932,615.75	**0.0000**	**0.00**	0.00
Cap72	977,799.4	977,799.4	977,799.4	**0.0000**	**0.00**	977,799.4	977,799.4	977,799.4	**0.0000**	**0.00**	0.00
Cap73	1,010,641.45	1,010,641.4500	1,010,641.45	**0.0000**	**0.00**	1,010,641.45	1,010,641.45	1,010,641.45	**0.0000**	**0.00**	0.00
Cap74	1,034,976.975	1,034,976.9750	1,034,976.975	**0.0000**	**0.00**	1,034,976.975	1,034,976.975	1,034,976.975	**0.0000**	**0.00**	0.00
Cap101	796,648.4375	796,648.4375	796,648.4375	**0.0000**	**0.00**	796,648.4375	796,648.4375	796,648.4375	**0.0000**	**0.00**	0.00
Cap102	854,704.2	854,704.2000	854,704.2	**0.0000**	**0.00**	854,704.2	854,704.2	854,704.2	**0.0000**	**0.00**	0.00
Cap103	893,782.1125	893,782.1125	893,782.1125	**0.0000**	**0.00**	893,782.1125	893,782.1125	893,782.1125	**0.0000**	**0.00**	0.00
Cap104	928,941.75	928,941.75	928,941.75	**0.0000**	**0.00**	928,941.75	928,941.75	928,941.75	**0.0000**	**0.00**	0.00
Cap131	794,299.85	802,709.225	798,355.4917	0.6196	2337.64	793,439.5625	793,439.5625	793,439.5625	**0.0000**	**0.00**	0.62
Cap132	851,495.325	854,704.2	852,300.2575	0.0945	813.37	851,495.325	851,495.325	851,495.325	**0.0000**	**0.00**	0.09
Cap133	893,076.7125	894,095.7625	893,352.4167	**0.0309**	359.03	894,095.7625	894,752.025	894,161.3888	0.1215	**200.24**	-0.09
Cap134	928,941.75	928,941.75	928,941.75	**0.0000**	**0.00**	928,941.75	928,941.75	928,941.75	**0.0000**	**0.00**	0.00
CapA	17,156,454.48	17,420,032.38	17,182,558.16	**0.1522**	**74782.61**	17,180,539.56	18,030,263.31	17,664,663.43	2.9622	236,833.5	-2.81
CapB	13,205,522.04	13,683,628.78	13,407,728.05	3.3027	109,738.5	13,100,041.02	13,476,652.7	13,304,594.27	**2.5081**	**91,430.13**	0.77
CapC	11,834,640.02	12,203,264.48	12,045,991.08	4.6968	95,778.78	11,535,255.5102	11,867,887.0012	11,802,532.8641	**2.5800**	**82,312.70**	2.02

Table 12. Comparison of IBPSO and binABC in terms of solution quality and robustness.

Problem	IBPSO					binABC					
	Best	Worst	Mean	GAP	Std. dev.	Best	Worst	Mean	GAP	Std. dev.	CR
Cap71	932,615.75	934,199.1375	932,964.7188	0.0374	587.49	932,615.75	932,615.75	932,615.75	**0.0000**	**0.00**	0.04
Cap72	977,799.4	983,122.2375	980,486.9238	0.2749	1844.64	977,799.4	977,799.4	977,799.4	**0.0000**	**0.00**	0.27
Cap73	1,010,641.45	1,014,917.6	1,012,642.103	0.1980	1513.78	1,010,641.45	1,010,641.45	1,010,641.45	**0.0000**	**0.00**	0.20
Cap74	1,034,976.975	1,045,383.788	1,039,149.436	0.4031	4426.67	1,034,976.975	1,034,976.975	1,034,976.975	**0.0000**	**0.00**	0.40
Cap101	796,648.4375	809,077.4875	801,403.0875	0.5968	3799.52	796,648.4375	796,648.4375	796,648.4375	**0.0000**	**0.00**	0.59
Cap102	856,660.0125	865,438.2	860,957.6625	0.7317	3249.38	854,704.2	854,704.2	854,704.2	**0.0000**	**0.00**	0.73
Cap103	894,008.1375	909,765.25	899,511.3063	0.6410	4978.98	893,782.1125	893,782.1125	893,782.1125	**0.0000**	**0.00**	0.64
Cap104	928,941.75	964,540.85	938,197.2625	0.9964	10,845.26	928,941.75	928,941.75	928,941.75	**0.0000**	**0.00**	0.99
Cap131	806,761.2875	821,236.3625	812,669.325	2.4236	4244.29	793,439.5625	793,439.5625	793,439.5625	**0.0000**	**0.00**	2.37
Cap132	865,407.5125	901,297.475	882,160.8613	3.6014	11,569.02	851,495.325	851,495.325	851,495.325	**0.0000**	**0.00**	3.48
Cap133	918,298.525	966,072.7125	940,076.1188	5.2626	14,905.27	894,095.7625	894,752.025	894,161.3888	**0.1215**	**200.24**	4.88
Cap134	973,744.7875	1,027,632.238	999,855.6225	7.6338	15,788.86	928,941.75	928,941.75	928,941.75	**0.0000**	**0.00**	7.09
CapA	35,704,093	45,511,134.48	40,812,839.67	137.8862	3,357,138.19	17,180,539.56	18,030,263.31	17,664,663.43	**2.9622**	**236,833.5**	56.72
CapB	17,971,425.35	22,154,126.28	20,152,622.54	55.2701	1,406,575.7	13,100,041.02	13,476,652.7	13,304,594.27	**2.5081**	**91,430.13**	33.98
CapC	14,564,601.36	18,555,781.02	16,747,099.16	45.5561	1,245,252.2	11,535,255.5102	11,867,887.0012	11,802,532.8641	**2.5800**	**82,312.7**	29.52

7. Results and discussion

In this paper, we propose a swarm intelligence-based optimizer called the binABC for solving binary optimization problems. The performance and accuracy of the proposed method are investigated and compared with the BPSO, DisABC, and IBPSO algorithms on UFLPs. The problems can be divided into 4 classes: small-sized (Cap71–74), medium-sized (Cap101–104), large-sized (Cap131–134), and very large-sized (CapA, CapB, and CapC). The proposed method has lower GAP values and standard deviations for the small-, medium-, and large-sized problems because the intensification (local search) of the binABC is very good. In BPSO, all of the solutions are reconstructed for the agents at each iteration. This behavior in the BPSO algorithm increases the diversification (global search) in the population but decreases the intensification of the population. Therefore, while BPSO is better than the DisABC and binABC algorithms for very large-sized problems, the DisABC and binABC are better than BPSO for small-, medium-, and large-sized problems. When the dimensionality of the problem increases, the accuracy of the DisABC algorithm is decreased, because the candidate food source position inherits too much from the previous food source positions, and the diversification in the population of the DisABC is decreased. The experimental results also show that the binABC algorithm is a good local searcher because only one parameter is updated for each solution at each iteration and global searcher, because if a solution could not be improved a certain time, the employed bee becomes a scout bee and the global search is thus provided in the binABC algorithm.

8. Conclusion and future work

This paper proposes a new binary version of the ABC (binABC) algorithm that works on binary-structured solution space. The key element of moving on the binary solution space is an XOR logic operator. In the binABC algorithm, the feasible food source positions are described on the binary solution space, the position of food sources consist of 0 and 1 logic values, and logic values 1 and 0 are to be opened and closed in the facility, respectively, for UFLPs. This solution representation can be extended to other binary optimization problems in the same way. According to experimental results, the proposed method is very effective for binary optimization and is an alternative tool for solving discrete binary optimization problems because it has a simple conceptual structure and only one peculiar control parameter (limit). Future works include adapting and implementing the binABC algorithm to solve different binary optimization problems such as feature selection for classification with support vector machines or artificial neural networks and swarm-based classification methods.

Acknowledgment

The authors wish to thank the 5 anonymous reviewers for their valuable suggestions and contributions. The authors also thank the Selçuk University Scientific Research Project Coordinatorship and the Scientific and Technological Research Council of Turkey for their institutional support.

References

[1] D. Karaboga, "An idea based on honey bee swarm for numerical optimization", Technical Report, Computer Engineering Department, Engineering Faculty, Erciyes University, available at http://mf.erciyes.edu.tr/abc/pub/tr06_2005.pdf, 2005.

[2] D. Karaboga, B. Basturk, "A powerful and efficient algorithm for numerical function optimization", Journal of Global Optimization, Vol. 39, pp. 459–471, 2007.

[3] D. Karaboga, B. Basturk, "Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems", Proceedings of the 12th International Fuzzy Systems Association World Congress on Foundations of Fuzzy Logic and Soft Computing, pp. 789–798, 2007.

[4] D. Karaboga, B. Akay, "A modified artificial bee colony (ABC) algorithm for constrained optimization problems", Applied Soft Computing, Vol. 11, pp. 3021–3031, 2011.

[5] D. Karaboga, B. Basturk, "On the performance of artificial bee colony (ABC) algorithm", Applied Soft Computing, Vol. 8, pp. 687–697, 2008.

[6] G. Zhu, S. Kwong, "Gbest-guided artificial bee colony algorithm for numerical function algorithm", Applied Mathematics and Computation, Vol. 217, pp. 3166–3173, 2010.

[7] B. Alatas, "Chaotic bee colony algorithms for global numerical optimization", Expert Systems with Applications, Vol. 37, pp. 5682–5687, 2010.

[8] M.H. Kashan, N. Nahavandi, A.H. Kashan, "DisABC: a new artificial bee colony algorithm for binary optimization", Applied Soft Computing, Vol. 12, pp. 342–352, 2011.

[9] D. Karaboga, B. Gorkemli, "A combinatorial artificial bee colony algorithm for traveling salesman problem", International Symposium on Innovations in Intelligent Systems and Applications, pp. 50–53, 2011.

[10] M.S. Kıran, H. İşcan, M. Gündüz, "The analysis of discrete artificial bee colony algorithm with neighborhood operator on traveling salesman problem", Neural Computing and Applications, Vol. 23, pp. 9–21, 2012.

[11] M.S. Kıran, M. Gündüz, "A novel artificial bee colony-based algorithm for solving the numerical optimization problems", International Journal of Innovative Computing, Information & Control, Vol. 8, pp. 6107–6122, 2012.

[12] B. Akay, D. Karaboga, "A modified artificial bee colony algorithm for real parameter optimization", Information Sciences, Vol. 192, pp. 120–142, 2012.

[13] D.J. Mala, V. Mohan, M. Kamalapriya, "Automated software test suite optimisation framework – an artificial bee colony optimisation-based approach", IET Software, Vol. 4, pp. 334–348, 2010.

[14] D. Karaboga, C. Ozturk, "Neural networks training by artificial bee colony algorithm on pattern classification", Neural Network World, Vol. 19, pp. 279–292, 2009.

[15] D. Karaboga, C. Ozturk, "A novel clustering approach: artificial bee colony (ABC) algorithm", Applied Soft Computing, Vol. 11, pp. 652–657, 2011.

[16] C. Ozturk, D. Karaboga, B. Gorkemli, "Probabilistic dynamic deployment of wireless sensor networks by artificial bee colony algorithm", Sensors, Vol. 11, pp. 6056–6065, 2011.

[17] C. Öztürk, D. Karaboğa, B. Görkemli, "Artificial bee colony algorithm for dynamic deployment of wireless sensor networks", Turkish Journal of Electrical Engineering & Computer Sciences, Vol. 20, pp. 255–262, 2012.

[18] D. Karaboga, S. Okdem, C. Ozturk, "Cluster based wireless sensor network routing using artificial bee colony algorithm", Wireless Networks, Vol. 18, pp. 735–747, 2012.

[19] D. Karaboga, C. Ozturk, N. Karaboga, B. Gorkemli, "Artificial bee colony programming for symbolic regression", Information Sciences, Vol. 209, pp. 1–15, 2012.

[20] A. Singh, "An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem", Applied Soft Computing, Vol. 9, pp. 625–631, 2009.

[21] N. Karaboğa, M.B. Çetinkaya, "A novel and efficient algorithm for adaptive filtering: artificial bee colony algorithm", Turkish Journal of Electrical Engineering & Computer Sciences, Vol. 19, pp. 175–190, 2011.

[22] N. Taşpınar, D. Karaboğa, M. Yıldırım, B. Akay, "PAPR reduction using artificial bee colony in OFDM systems", Turkish Journal of Electrical Engineering & Computer Sciences, Vol. 19, pp. 47–58, 2011.

[23] D. Karaboga, B. Gorkemli, C. Ozturk, N. Karaboga, "A comprehensive survey: artificial bee colony (ABC) algorithm and applications", Artificial Intelligence Review, DOI 10.1007/s10462-012-9328-0, 2012.

[24] A.R. Guner, M. Sevkli, "A discrete particle swarm optimization algorithm for uncapacitated facility location problem", Journal of Artificial Evolution and Applications, Vol. 2008, pp. 1–9, 2008.

[25] J. Kennedy, R. Eberhart, "A discrete binary version of the particle swarm algorithm", IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, Vol. 5, pp. 4101–4109, 1997.

[26] X. Yuan, H. Nie, A. Su, L. Wang, Y. Yuan, "An improved binary particle swarm optimization for unit commitment problem", Expert System with Applications, Vol. 39, pp. 8049–8055, 2009.

[27] Y. Shi, R.C. Eberhart, "A modified particle swarm optimizer", Proceedings of the IEEE International Conference on Evolutionary Computation, pp. 69–73, 1998.

[28] R.D. Galvâo, L.A. Raggi, "A method for solving to optimality uncapacitated location problems", Annals of Operations Research, Vol. 18, pp. 225–244, 1989.

[29] J. Krarup, P.M. Pruzan, "The simple plant location problem: survey and synthesis", European Journal of Operations Research, Vol. 12, pp. 36–81, 1983.

[30] M.S. Daskin, L.V. Snyder, R.T. Berger, "Facility location in supply chain design", Lehigh University, Working Paper No.: 03-010, available at http://www.lehigh.edu/∼lvs2/Papers/facil-loc-sc.pdf, 2003.

[31] G. Cornuéjols, G.L. Nemhauser, L.A Wolsey, "The uncapacitated facility location problem", Lecture Notes in Artificial Intelligence, Vol. 1865, pp. 119–171, 1990.

[32] K. Holmberg, "Exact solution methods for uncapacitated location problems with convex transportation costs", European Journal of Operational Research, Vol. 114, pp. 127–140, 1999.

[33] J. Barcelo, Å. Hellfjord, E. Fernandes, K. Jörnsten, "Lagrangian relaxation and constraint generation procedures for capacitated plant location problems with single sourcing", OR Spectrum, Vol. 12, pp. 78–79, 1990.

[34] D. Erlenkotter, "A dual-based procedure for uncapacitated facility location", Operations Research, Vol. 26, pp. 992–1009, 1978.

[35] J.H. Jaramillo, J. Bhadury, R. Batta, "On the use of genetic algorithms to solve location problems", Computers & Operations Research, Vol. 29, pp. 761–779, 2002.

[36] K.S. Al-Sultan, M.A. Al-Fawzan, "A tabu search approach to the uncapacitated facility location problem", Annals of Operations Research, Vol. 86, pp. 91–103, 1999.

[37] M. Sun, "Solving the uncapacitated facility location problem using tabu search", Computers & Operations Research, Vol. 33, pp. 2563–2589, 2006.

[38] M. Sevkli, A.R. Guner, "A continuous particle swarm optimization algorithm for uncapacitated facility location problem", Proceedings of the 5th International Workshop on Ant Colony Optimization and Swarm Intelligence, Brussels, Belgium, pp. 316–323, 2006.

[39] J. Byrka, K. Aardal, "An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem", SIAM Journal on Computing, Vol. 39 pp. 29–43, 2007.

[40] J.E. Beasley, "OR-Library: distributing test problems by electronic mail", Journal of the Operational Research Society, Vol. 41, pp. 1069–1072, 1990.

[41] D. Karaboga, B. Akay, "A comparative study of artificial bee colony algorithm", Applied Mathematics and Computation, Vol. 214, pp. 108–132, 2009.