



A binary social spider algorithm for continuous optimization task

Emine Baş¹ · Erkan Ülker²

Published online: 30 January 2020
© Springer-Verlag GmbH Germany, part of Springer Nature 2020

Abstract

The social spider algorithm (SSA) is a new heuristic algorithm created on spider behaviors. The original study of this algorithm was proposed to solve continuous problems. In this paper, the binary version of SSA (binary SSA) is introduced to solve binary problems. Currently, there is insufficient focus on the binary version of SSA in the literature. The main part of the binary version is at the transfer function. The transfer function is responsible for mapping continuous search space to discrete search space. In this study, four of the transfer functions divided into two families, S-shaped and V-shaped, are evaluated. Thus, four different variations of binary SSA are formed as binary SSA-Tanh, binary SSA-Sigm, binary SSA-MSigm and binary SSA-Arctan. Two different techniques (SimSSA and LogicSSA) are developed at the candidate solution production schema in binary SSA. SimSSA is used to measure similarities between two binary solutions. With SimSSA, binary SSA's ability to discover new points in search space has been increased. Thus, binary SSA is able to find global optimum instead of local optimums. LogicSSA which is inspired by the logic gates and a popular method in recent years has been used to avoid local minima traps. By these two techniques, the exploration and exploitation capabilities of binary SSA in the binary search space are improved. Eighteen unimodal and multimodal standard benchmark optimization functions are employed to evaluate variations of binary SSA. To select the best variations of binary SSA, a comparative study is presented. The Wilcoxon signed-rank test has applied to the experimental results of variations of binary SSA. Compared to well-known evolutionary and recently developed methods in the literature, the variations of binary SSA performance is quite good. In particular, binary SSA-Tanh and binary SSA-Arctan variations of binary SSA showed superior performance.

Keywords Binary optimization · Social spider algorithm · Transfer function

1 Introduction

Evolutionary computation has become an attractively efficient device of optimization for rapidly increasing complex modern optimization problems. Evolutionary computation based on natural facts can be separated into two important groups. These are evolutionary algorithms

(EAs) and swarm intelligence-based algorithms. EAs include very successful methods that are mainly created by inspirations from nature. There are a lot of EAs which solve real-world problems and global optimization problems. The main ones of these are the genetic algorithm (GA), genetic programming (GP), evolutionary strategies (ES) and differential evolution (DE). These algorithms have had very successful results, especially in solving convex optimization problems (Talbi 2009; Mallipeddi et al. 2011).

For the last 20 years, swarm intelligence-based algorithms, which are based on a new evolutionary computation method, attract attention. The term of a swarm expresses a community that consists of individuals who are in communication with each other. Swarm intelligence-based algorithms study main social animal and insect behaviors for solving problems. These algorithms imitate the behaviors of ant, fish, bird, bee, bacteria, butterfly, etc. Thus, the problems which seem hard are able to be solved

Communicated by V. Loia.

✉ Emine Baş
emineozcan@selcuk.edu.tr
Erkan Ülker
eulker@ktun.edu.tr

¹ Kulu Vocational School, Selçuk University, 42075 Konya, Turkey

² Department of Computer Engineering, Faculty of Engineering and Nature Sciences, Konya Technical University, 42075 Konya, Turkey

(Parpinelli and Lopes 2011; Yu and Li 2015). Three well-known pioneers of this area are: particle swarm optimization (PSO), ant colony optimization (ACO) (Kennedy and Eberhart 1995; Dorigo 1990) and artificial bee colony (ABC) optimization algorithm (Karaboga 2005). Particle swarm optimization is an optimization method, which is developed by Kennedy and Eberhart (1995) inspired by flocking fish and insects. ACO is an algorithm inspired by ants' behaviors. The aim is to find the shortest path to food sources for an ant colony. Karaboga has designed an artificial bee colony (ABC) optimization algorithm inspired by bees' behaviors (Omkar et al. 2011).

1.1 Binary optimization problem (BOP)

The binary optimization problem (BOP) is shown as a binary-based problem space that represents an important class of the combinatorial optimization problems (Rizk-Allah et al. 2018). In continuous optimization, search agents take continuous values in search space, while in binary optimization search agents in the search space take $\{0, 1\}$ values. "0" represents the absence, and "1" represents the presence. Many problems can be solved in a binary space by using these two values in the search space. Some algorithms are capable of solving problems with continuous search spaces, while some problems have discrete search spaces (Mirjalili and Lewis 2013; Kennedy and Eberhart 1997; Rashedi et al. 2009). BOP has many applications. They need binary algorithms for their solution. These include facility location (including emergency vehicles, health centers and commercial bank branches) and scheduling tasks (including budgeting, flexible manufacturing systems, telecommunications, mass transit services and wind turbine placement) (Prescilla and Immanuel 2013; Korkmaz et al. 2017; Beskirli et al. 2018). Also, the BOP is used in the solution of well-known NP-hard problems [including knapsack problem, resource allocation problem, dimensionality reduction, feature selection, network optimization, unit commitment and cell formation (Rizk-Allah and Hassanien 2018; Rizk-Allah 2014; Fan et al. 2013; Pal and Maiti 2010; Babaoglu et al. 2010; Qiao et al. 2006; Emary et al. 2016)]. In the literature, many traditional methods [including relaxation methods, Lagrangian techniques, branch-and-bound methods, reduction schemes and integer programming (Rizk-Allah et al. 2018)] have been proposed to solve the BOP. Although these methods perform well in small-scale problems, they do not perform well in large-scale problem solutions. Although there are many methods for the solution of BOPs, the solution with meta-heuristic algorithms has given us many advantages. In particular, it has shortened the solution time of large-scale problems. Due to these limitations in the deterministic methods, more and

more are becoming interested in the meta-heuristic algorithms inspired by specific phenomena (Shukla and Nanda 2018; Pereira et al. 2014; Rizk-Allah et al. 2018; Mirjalili and Lewis 2013; Kennedy and Eberhart 1997; Rashedi et al. 2009; Prescilla and Immanuel 2013; Korkmaz et al. 2017; Beskirli et al. 2018; Rizk-Allah and Hassanien 2018; Babaoglu et al. 2010; Emary et al. 2016; Ling et al. 2010a, b). There are different methods to develop the binary version of a continuous heuristic algorithm while preserving the concepts of the search process (Mirjalili and Lewis 2013). For instance, Ling et al. developed a probability estimation operator in order to solve binary problems by DE (BDE) (Ling et al. 2010a). The binary magnetic optimization algorithm (BMOA) and binary gravitational search algorithm (BGSA) have been proposed utilizing transfer functions and position updating rules (Mirjalili and Mohd Hashim 2012; Rashedi et al. 2009). Binary harmony search (BHS) algorithm has been employed a set of harmony search considerations and pitch adjustment rules (Ling et al. 2010b). Kennedy and Eberhart (1997) proposed binary versions of PSO. They have used two different components for BPSO (a new transfer function and a different position updating procedure). Rizk-Allah and Hassanien (2018) proposed a binary version of the bat algorithm (BBA). Cuevas et al. (2013) have developed social spider optimization (SSO). Shukla and Nanda (2018) developed a binary social spider optimization algorithm (BSSO) for high-dimensional data sets. Rizk-Allah et al. (2018) proposed binary salp swarm algorithm (BSalpSA) and Rizk-Allah (2018) proposed binary sine cosine algorithm (BSCA).

Yu and Li (2015) have offered social spider algorithm (SSA) which is configured according to social spider behaviors. Some spider species, which generally live solitary, can live as colonies. Spiders live in colonies can communicate with each other by a web structure. The spiders enable this communication with the vibration that they produce on the web. They can make random walks to each other by the vibration information they share (Yu and Li 2015). There are not many studies on SSA in the literature. El-Bages and Elsayed have solved the static transmission expansion planning problem by using SSA. In the proposed method, the DC power flow subproblem is solved by developing an SSA-based web and adding potential solutions to the result (El-Bages and Elsayed 2017). Yu and Li have solved economic load dispatch (ELD) formulation by developing an SSA-based new approach. ELD is one of the essential components in power system control and operation (Yu and Li 2016). Mousa and Bentahar have adapted a QoS-aware web service selection process to the SSA approach (Mousa and Bentahar 2016). Elsayed et al. (2016) have solved the non-convex economic load dispatch problem with an SSA-based approach.

Social spider optimization (SSO) is another heuristic algorithm that is very similar to the SSA algorithm, but totally different from it. Cuevas et al. (2013) have developed SSO algorithm by inspiring from spider behaviors. In SSO, there are two different searching agents (spider) in the search space of SSO. These are male and female spiders. Depending upon the gender, each individual operates a range of different evolutionary operators that imitates various collaboration behaviors that typically exist in colonies. In SSO, the vibration structure is very different. There are three special vibration types in SSO approach. Vibci: They are perceived by the individual i (S_i) as a result of the information transmitted by the member c (S_c) who is an individual that has two important characteristics: It is the nearest member to i and possesses a higher weight ($w_c > w_i$). Vibbi: They are perceived by the individual i as a result of the information transmitted by the member b (S_b), with b being the individual holding the best weight (best fitness value). Vibfi: They are perceived by the individual i (S_i) as a result of the information transmitted by the member f (S_f), with f being the nearest female individual to i . There are also various different studies in the literature with SSO. Shukla and Nanda (2018) have developed a parallel social spider clustering algorithm for high-dimensional data sets and a BSSO algorithm for unsupervised band selection in compressed hyperspectral images. Pereira et al. (2014) have developed a SSO-based artificial neural networks training and applied its applications for Parkinson's disease identification. Sun et al. (2017) have developed the hybrid SSO algorithms and applied it for the estimation of thermophysical properties of phase change material.

1.2 Binary SSA for continuous optimization task

The spider social algorithm is a heuristic algorithm developed to solve continuous optimization problems. The aim of this study is to introduce a binary social spider algorithm (binary SSA) for solving general BOPs. SSA is chosen in this study because it is a new heuristic algorithm and there is not enough focus on binary SSA in the literature. Furthermore, there is no study in the literature on the performance of binary SSA with respect to different transfer functions. The main part of the binary optimization is at transfer function. SSA is modified again according to four different transfer functions for mapping the continuous search space to the binary search space. In this paper, four binary variations of the binary SSA (i.e., binary SSA-Tanh, binary SSA-Sigm, binary SSA-MSigm and binary SSA-Arctan) are proposed based on four transfer functions. In order to ensure the balance between exploration and exploitation, two different techniques (similarity measurement and logic gate techniques) are developed at the

candidate solution production schema in the variations of the binary SSA. The similarity measurement technique is named as SimSSA, and the logic gate technique is named as LogicSSA. Thus, the variations of binary SSA based on SimSSA and LogicSSA is developed in this study. The logic gate technique has a newly developed method in recent years and preferred for the developed binary SSA. The logic gate technique provides a strong local search capacity. Logic gates are very suitable for binary optimizations due to their structure. Input and output values of logic gates are 0 or 1 value. In binary optimizations, the search space consists of 0 or 1 value. 0 means the absence of a value, and 1 indicates its existence. So, it is appropriate to use logic gates in the search space. The similarity measurement technique is used to measure the similarity between two different binary structures, and the exploration ability of binary SSA is increased with the similarity measurement technique. This ensures that global optimum is found in the search space instead of local optimums. The convergence speed of binary SSA is also increased by SimSSA and LogicSSA.

The primary contributions of this paper are as follows:

- (a) We propose a binary SSA to solve general BOPs. The SSA has modified again according to different transfer functions for mapping the continuous search space to the binary search space.
- (b) Four different variations of binary SSA (i.e., binary SSA-Tanh, binary SSA-Sigm, binary SSA-MSigm and binary SSA-Arctan) are developed with four different transfer functions.
- (c) In order to ensure the balance between exploration and exploitation, two different techniques (SimSSA and LogicSSA) are developed at the candidate solution production stage in the variations of the binary SSA. There are advantages that both techniques offer different types. For example, LogicSSA has a strong local search capacity around the current spider, and SimSSA is good at discovering the new points on the solution space. So new candidate solutions (new spiders) are produced by using similarity and logic gates and more efficient individuals are obtained by comparing with the current individuals.
- (d) The variations of binary SSA based on SimSSA and LogicSSA have been tested on various unimodal and multimodal benchmark functions. Thus, the performances between variations of binary SSA are compared.

The rest of the paper is organized as follows: SSA is presented in Sect. 2, the variations of binary SSA based on SimSSA and LogicSSA are studied in Sect. 3 and the variations of binary SSA are tested by various unimodal

and multimodal benchmark functions in Sect. 4 and the obtained results are compared with each other and well-known heuristic methods and new methods in the literature. The results are evaluated.

2 Social spider algorithm (SSA)

Social spider algorithm (SSA) is a heuristic algorithm that is created by imitating spiders’ behaviors in nature. SSA is created for organizing searching space of optimization problems as a high-dimensional spider web. In SSA, spiders are SSA agents that perform optimization. At the beginning of the algorithm, a predetermined number of spiders are placed on the web. Each spider on the web has a memory. Spider information has stored this memory (the position of spider, the movement of spider, the dimension mask of spider, etc.) (Yu and Li 2015). The locations of spiders are initially set at random. Each spider produces a vibration when it moves from a position to a different location. The severity of the vibration is related to the fitness value. The vibration is the most important feature that distinguishes SSA from other optimization algorithms. Vibration can spread over the web, and other spiders on the web can feel it. Thus, other spiders on the web can obtain the personal information of the spider. Equation 1 is used to calculate the vibration intensity (Yu and Li 2015).

$$I(P_g, P_g, t) = \log\left(\frac{1}{f(p_g) - C}\right) + 1 \tag{1}$$

where $I(P_g, P_g, t)$ is the vibration value produced by the spider in the source position at time t . “ g ” represents a spider. $P_g(t)$ defines the position of spider g at time t and it is shown simply as P_g if the time argument is t . $f(P_g)$ represents the spider of fitness value in its current position. C is a confidently small constant such that all possible fitness values are larger than C value for minimization problems. Logarithms are used for operations with numbers that are too large or too small.

$$\text{Dis}(P_g, P_b) = P_g - P_b \tag{2}$$

$\text{Dis}(P_g, P_b)$ is showing the distance between spider g and spider b . This distance is calculated by Eq. 2 (Yu and Li 2015). Manhattan distance structure is used when calculating this distance. The vibration attenuation over distance is calculated by Eq. 3 (Yu and Li 2015). For vibration attenuation, the vibration value produced by the spider in the source position and distance are two important properties.

$$I(P_g, P_b, t) = I(P_g, P_g, t) \times \exp\left(-\frac{\text{Dis}(P_g, P_b)}{\bar{\sigma} \times r_a}\right) \tag{3}$$

where $I(P_g, P_b, t)$ is the felt value of the spider’s vibration in “ g ” point, by the spider in “ b ” point. $P_g(t)$ or simply P_g represents the position of spider g at time t . $P_b(t)$ or simply P_b represents the position of spider b at time t . $I(P_g, P_g, t)$ is the vibration value produced by the spider in the “ g ” position at time t . $\bar{\sigma}$ represents the mean of the standard deviation of the positions of all spiders in each dimension. r_a is represented as a user-controlled parameter $r_a \in (0, \infty)$. This parameter controls the attenuation rate of the vibration intensity over the distance. The larger the r_a is, the weaker the attenuation imposed on the vibration (Yu and Li 2015). The value of r_a is selected from the set $\{1/10, 1/5, 1/4, 1/3, 1/2, 1, 2, 3, 4, 5, 10\}$. Yu and Li (2015) have used for five 10-dimensional benchmark functions to investigate the impact of this parameter on the performance of SSA. According to their results, $r_a = 1$ is determined.

In the initialization phase, each spider is assigned a fixed size memory for storing spider information. The positions of spiders are randomly generated in the search space. Fitness values of the spiders calculated and stored. All spiders produce vibration using their position (Eq. 1). The algorithm uses vibrations to activate the propagation process (Eq. 3). “ V ” represents spiders’ vibrations. Each spider receives the vibrations generated by other spiders. A spider selects the strongest of these vibration values. This value shows as V_g^{best} , g represents a spider. Spider g stores the target vibration in the memory as V_g^{tar} . Each spider g compares V_g^{best} and V_g^{tar} values. If the intensity of the V_g^{best} is greater than V_g^{tar} , the V_g^{tar} is changed to as V_g^{best} . The random walk of each spider is calculated by Eq. 4 (Yu and Li 2015).

$$P_g(t + 1) = P_g + (P_g - P_g(t - 1)) \times r + (P_g^{\text{follow}} - P_g) \odot R \tag{4}$$

where R is a vector of random float point numbers generated from zero to one uniformly. \odot denotes element-wise multiplication. Spider g first moves along its previous direction, which is the direction of movement in the previous iteration. The distance along this direction is a random portion of the previous movement. Then spider g approaches P_g^{follow} along each dimension with random factors generated in $(0, 1)$. P_g^{follow} represents the following position. Each spider decides the following position (P_g^{follow}) by looking at the dimension mask.

The dimension mask is determined so that a random walk can be performed toward the V_g^{tar} . The dimension mask is a $\{0, 1\}$ binary vector of length D , and D is the dimension of the optimization problem. Figure 1 shows the dimension mask for SSA (Population size \times Dimension).

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |

Fig. 1 Dimension mask for SSA (Population size x Dimension)

The dimension mask in SSA is a structure different from the mutation operator of the genetic algorithm. In the genetic algorithm, the mutation operator is used to prevent new solutions from copying the previous solution and to reach the result faster (Holland 1975; Kurt and Sematay 2001). For example, in a sequence using a binary encoding, a new sequence is obtained by changing the element value randomly selected by the mutation operator to 1 if it is 0 or 1 if it is 0. In the SSA, the dimension mask guides the random walk phase. Dimension mask is not an individual, candidate solution or gene. Using the dimension mask, spiders move toward the V_g^{tar} or a random spider during the random walk. Each spider decides the following position (P_g^{follow}) by looking at the dimension mask (in Eq. 5). With the dimension mask, SSA achieves both global search capability and local search capability. In SSA, a spider obtains local search capability by performing move toward the V_g^{tar} and global search capability by performing move toward a random spider.

In SSA, all bits of the dimension mask are zero at first. In each iteration, spiders have a probability of $1 - p_c^{cs}$ to change its dimension mask. $p_c \in (0, 1)$ is a user-defined attribute that describes the probability of changing mask. If the dimension mask is decided to be changed, each bit of the vector has a probability of p_m . p_m is also a user-controlled parameter defined in $(0, 1)$. The values of p_c and p_m are selected from the set $\{0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99\}$. Yu and Li are used five 10-dimensional benchmark functions to investigate the impact of these parameters on the performance of SSA. According to their results, p_c and p_m values are determined as 0.7 and 0.1, respectively. cs is the number of iterations that the spider has last changed its target vibration. Each bit of the dimension mask is changed independently. The new dimension mask does not have any correlation with the previous dimension mask.

After the dimension mask is determined, a new following position P_g^{follow} is generated based on the dimension mask. P_g^{follow} value is calculated by Eq. 5 (Yu and Li 2015). If the bit value of the dimension mask is 0, it moves to the position of the target spider in the memory of the spider or to the position of a random spider.

$$P_{g,i}^{follow} = \begin{cases} P_{g,i}^{tar}, & dimension_mask_{g,i} = 0 \\ P_{g,i}^r, & dimension_mask_{g,i} = 1 \end{cases} \quad (5)$$

where r is a random integer value generated in $[1, \text{population size}]$ and $dimension_mask_{g,i}$ stands for the i th dimension of the dimension mask of spider g .

After each spider performs a random walk, the spiders may move out of the web. They violate the constraint of the optimization problem. Equation 6 is used to prevent population members from moving out of the search space.

$$P_{g,i}(t + 1) = \begin{cases} (\bar{x}_i - P_{g,i}) \times r & \text{if } P_{g,i}(t + 1) > \bar{x}_i \\ (P_{g,i} - \underline{x}_i) \times r & \text{if } P_{g,i}(t + 1) < \underline{x}_i, \end{cases} \quad (6)$$

where \bar{x}_i shows upper border and \underline{x}_i shows a lower border. $P_{g,i}(t + 1)$ defines the position of a spider g for the i th dimension at time $t + 1$. r is a random integer value generated in $[1, \text{population size}]$.

Flowchart for SSA is shown in Fig. 2. The work steps of SSA are shown in Fig. 3.

3 A binary social spider algorithm (binary SSA)

In general, there are many problems interested in binary search spaces, such as feature selection and diminishing dimensionality. In addition, algorithms that have continuous real search spaces can solve binary optimization problems (BOPs) by converting variables into binary variables. Regardless of the type of BOPs, binary search

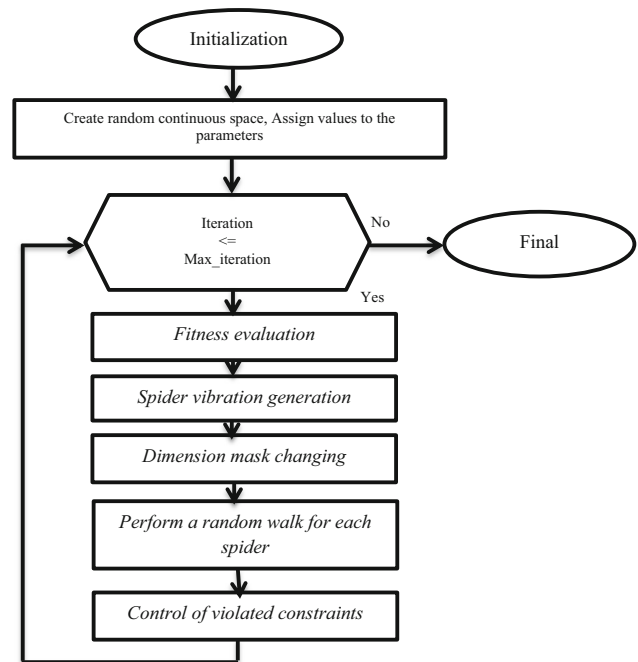


Fig. 2 Flowchart for SSA

- Step 1. Parameter setting:** Required constant parameter sets are adjusted. For example, lower and upper limits of variables, number of maximum iteration, etc.
- Step 2. Initialization:** Random starting position locations are determined for each spider in the swarm. Assign memory for them. The target vibration for each spider is assigned.
- Step 3. Evaluation:** Fitness value is calculated for each spider. The best value in the spider population is detected and saved.
- Step 4.** Vibration value is calculated for each spider calculated by using vibration Equations (1,2, and 3). The best value in the spider population is detected and saved.
- Step 5.** Vibration comparison: The received the best vibration value is compared to the target vibration value for each spider in population.
- Step 6.** Update the dimension mask.
- Step 7.** P^{follow} value is calculated by Equation 5
- Step 8. Movement:** Each spider moves to a new position by Equation 4 and violations are checked.
- Step 9. Repeat:** All criteria beginning from Step 3 are repeated until reaching stopping criteria.
- Step 10. Output:** Optimum outputs are obtained.

Fig. 3 Work steps of SSA

space has its own unique structure with some limitations. A binary search space can be considered as a hypercube. The agents (spiders) of a binary optimization algorithm can only shift to nearer and farther corners of the hypercube by flipping various numbers of bits (Kennedy and Eberhart 1997). Hence, in designing the binary version of SSA, the position updating process must be modified. In SSA, social spiders continually change their positions as a random position in the search space according to the lower and upper limit ranges. In binary optimization, the solution is restricted to the binary {0, 1} values.

It has been noticed that there is not enough focus on the binary version of SSA in the literature. The transfer functions, which are the main part of the binary transformation, are handled individually and their performance is not studied. There is no study on the performance of SSA to solve general BOPs. For these reasons, we have proposed a binary SSA to perform general binary tasks in this paper. We have obtained different binary SSA versions using various transfer functions and compared their performance. In binary SSA, two new different techniques (SimSSA and LogicSSA) are developed in the candidate solution production scheme to provide the balance between exploration and exploitation in search space. These are based on similarity measurement (SimSSA) and logical gate (LogicSSA) techniques. There are advantages offered by different types of both techniques. For example, LogicSSA is very powerful in local search and SimSSA is very good at discovering new solutions in solution space. Since both techniques have advantages offered in different types.

In the binary SSA, social spiders update their positions on the binary search space according to Eq. 4. In the binary SSA, the solution pool is represented by {0, 1} binary string values. The position of a spider is updating through 0–1 flipping operation. This flipping process is performed by a certain threshold value that is related to the transfer function. The transfer function is the most important part of converting to

binary form. A transfer function defines the probability of changing a position vector’s elements from 0 to 1 and vice versa (Kennedy and Eberhart 1997). There are many S-shaped and V-shaped transfer functions in the literature. Transfer function families of S-shaped and V-shaped are shown in Table 1. In this paper, two S-shaped (sigmoidal transfer function and modified sigmoidal transfer function) and two V-shaped (tangent hyperbolic transfer function and arctan transfer function) transfer functions are used. Thus, four different binary SSA variations such as binary SSA-Tanh, binary SSA-Sigm, binary SSA-MSigm and binary SSA-Arctan are formed. The transfer function to be selected for binary conversion affects the performance of the algorithm used in the BOPs. A transfer function should offer both a high probability and a small possibility of changing positions.

3.1 According to the transfer functions, the variations of binary SSA

3.1.1 Binary SSA-Tanh (tangent hyperbolic transfer function)

In binary SSA-Tanh, a tangent hyperbolic transfer function is used as the transfer function. The positions of the social

Table 1 S-shaped and V-shaped transfer functions (Mirjalili and Lewis 2013)

| S-shaped family | | V-shaped family | |
|-----------------|---------------------------------------|-----------------|--|
| Name | Transfer function | Name | Transfer function |
| S1 | $T(x) = \frac{1}{1+e^{-2x}}$ | V1 | $T(x) = \left \operatorname{erf}\left(\frac{\sqrt{\pi}}{2}x\right) \right $ |
| S2 | $T(x) = \frac{1}{1+e^{-x}}$ | V2 | $T(x) = \tanh(x) $ |
| S3 | $T(x) = \frac{1}{1+e^{-\frac{x}{2}}}$ | V3 | $T(x) = \left \frac{x}{\sqrt{1+x^2}} \right $ |
| S4 | $T(x) = \frac{1}{1+e^{-\frac{x}{3}}}$ | V4 | $T(x) = \left \frac{2}{\pi} \arctan\left(\frac{\pi}{2}x\right) \right $ |

spiders are converted into binary form according to the tangent hyperbolic transfer function. This function works as a threshold limit and converts positions between 0 and 1 values. The tangent hyperbolic transfer function is calculated by Eq. 7 (Rizk-Allah et al. 2018).

$$T(x_i^j(t+1)) = \text{Tanh}(x_i^j(t+1)) = \frac{e^{(2x_i^j(t+1))_{-1}}}{e^{(2x_i^j(t+1))_{+1}}} \tag{7}$$

$$\Delta_i^j(t+1) = \begin{cases} 1 & \text{if } T(x_i^j(t+1)) > \lambda_1 \\ 0 & \text{Otherwise} \end{cases}$$

where λ_1 is a random number in [0–1]. $x_i^j(t+1)$ is the position of spider i at iteration t in j th dimension.

3.1.2 Binary SSA-Sigm (sigmoidal transfer function)

In binary SSA-Sigm, a sigmoidal transfer function is used as the transfer function. The positions of the social spiders are converted into binary form according to the sigmoidal transfer function. This function works as a threshold limit and converts positions between 0 and 1 values. The sigmoidal transfer function is calculated by Eq. 8 (Rizk-Allah et al. 2018).

$$S(x_i^j(t+1)) = \text{Sig}(x_i^j(t+1)) = \frac{1}{1 + e^{-x_i^j(t+1)}} \tag{8}$$

$$\Delta_i^j(t+1) = \begin{cases} 1 & \text{if } S(x_i^j(t+1)) > \lambda_2 \\ 0 & \text{Otherwise} \end{cases}$$

where λ_2 is a random number in [0–1]. $x_i^j(t+1)$ is the position of spider i at iteration t in j th dimension.

3.1.3 Binary SSA-MSigm (modified sigmoidal transfer function)

In binary SSA-MSigm, a modified sigmoidal transfer function is used as the transfer function. The positions of the social spiders are converted into binary form according to the modified sigmoidal transfer function. This function works as a threshold limit and converts positions between 0 and 1 values. The modified sigmoidal transfer function is calculated by Eq. 9 (Rizk-Allah et al. 2018).

$$\text{MS}(x_i^j(t+1)) = \text{MSig}(x_i^j(t+1)) = \frac{1}{1 + e^{-10(x_i^j(t+1)-0.5)}} \tag{9}$$

$$\Delta_i^j(t+1) = \begin{cases} 1 & \text{if } \text{MS}(x_i^j(t+1)) > \lambda_3 \\ 0 & \text{Otherwise} \end{cases}$$

where λ_3 is a random number in [0–1]. $x_i^j(t+1)$ is the position of spider i at iteration t in j th dimension.

3.1.4 Binary SSA-Arctan (arctan transfer function)

In binary SSA-Arctan, an arctan transfer function is used as the transfer function. The positions of the social spiders are converted into binary form according to the arctan transfer function. This function works as a threshold limit and converts positions between 0 and 1 values. The arctan transfer function is calculated by Eq. 10 (Rizk-Allah et al. 2018).

$$A(x_i^j(t+1)) = A(x_i^j(t+1)) = \left| \frac{2}{\pi} \arctan\left(\frac{2}{\pi} x_i^j(t+1)\right) \right|$$

$$\Delta_i^j(t+1) = \begin{cases} 1 & \text{if } A(x_i^j(t+1)) > \lambda_4 \\ 0 & \text{Otherwise} \end{cases} \tag{10}$$

where λ_4 is a random number in [0–1]. $x_i^j(t+1)$ is the position of spider i at iteration t in j th dimension.

3.2 Similarity-based social spider algorithm (SimSSA)

3.2.1 Similarity measurement for binary structures

The similarity measurement technique is developed to measure the similarity between two different binary structures. There are 76 similarity measurement techniques for binary data (Qiao et al. 2006). One of the most commonly used general-purpose similarity measurement techniques is shown in Eq. 11 (Jaccard’s similarity measurement technique) (Çınar and Kiran 2018).

$$\text{Similarity}(X_i, Y_i) = \frac{m_{11}}{m_{01} + m_{10} + m_{11}} \tag{11}$$

where X_i and Y_i represent two binary structured individuals. “ D ” is the dimensionality of the problem for X_i and Y_i . $X_{id} \in \{0, 1\}$, $X_i = [x_{i1}, x_{i2}, \dots, x_{id}]$, $Y_{id} \in \{0, 1\}$, $Y_i = [y_{i1}, y_{i2}, \dots, y_{id}]$. In order to measure the similarity between X_i and Y_i , we compare them by means of their bit values. There are four possible cases, and they are shown in Table 2.

m_{11} indicates the number of bits when $x_{id} = 1$ in X_i and $y_{id} = 1$ in Y_i , where $i = 1, 2, \dots, D$ and D is the dimensionality of the problem, which is obtained as follows (Emary et al. 2016):

$$m_{11} = \sum_{d=1}^D I_d, \quad \text{where} \tag{12}$$

$$I_d = \begin{cases} 1, & \text{if } (x_{id} = 1 \text{ and } y_{id} = 1) \\ 0, & \text{otherwise} \end{cases}$$

m_{10} indicates the number of bits when $x_{id} = 1$ in X_i and $y_{id} = 0$ in Y_i , where $i = 1, 2, \dots, D$ and D is the

Table 2 Possible cases in similarity

| Cases | x_{id} | y_{id} | m_{11} | m_{10} | m_{01} | m_{00} |
|-------|----------|----------|----------|----------|----------|----------|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 1 |

dimensionality of the problem, which is obtained as follows (Emary et al. 2016):

$$m_{10} = \sum_{d=1}^D I_d, \text{ where} \tag{13}$$

$$I_d = \begin{cases} 1, & \text{if } (x_{id} = 1 \text{ and } y_{id} = 0) \\ 0, & \text{otherwise} \end{cases}$$

m_{01} indicates the number of bits when $x_{id} = 0$ in X_i and $y_{id} = 1$ in Y_i , where $i = 1, 2, \dots, D$ and D is the dimensionality of the problem, which is obtained as follows (Emary et al. 2016):

$$m_{01} = \sum_{d=1}^D I_d, \text{ where} \tag{14}$$

$$I_d = \begin{cases} 1, & \text{if } (x_{id} = 0 \text{ and } y_{id} = 1) \\ 0, & \text{otherwise} \end{cases}$$

m_{00} indicates the number of bits when $x_{id} = 0$ in X_i and $y_{id} = 0$ in Y_i , where $i = 1, 2, \dots, D$ and D is the dimensionality of the problem, which is obtained as follows (Emary et al. 2016):

$$m_{00} = \sum_{d=1}^D I_d, \text{ where} \tag{15}$$

$$I_d = \begin{cases} 1, & \text{if } (x_{id} = 0 \text{ and } y_{id} = 0) \\ 0, & \text{otherwise} \end{cases}$$

m_{00} is not used in Jaccard’s similarity measurement technique, because the value of 0 between two binary structures does not make sense in Jaccard’s similarity measurement technique.

By taking into account Jaccard’s coefficient of similarity, an intuitive measure of dissimilarity between X_i and Y_i , which defines how far apart X_i and Y_i are from each other, can be calculated as follows (Çınar and Kiran 2018):

$$\begin{aligned} \text{Dissimilarity}(X_i, Y_i) &= 1 - \text{Similarity}(X_i, Y_i) \\ &= 1 - \frac{m_{11}}{m_{01} + m_{10} + m_{11}} \end{aligned} \tag{16}$$

where dissimilarity (X_i, Y_i) is in the range of [0, 1].

3.2.2 Generating a new solution in SimSSA

Two different spider values are used to generate a new solution in SimSSA (present spider and best spider

(spider_{best} in spider population) or present spider and randomly selected neighbor spider). In order to generate a new solution in the binary search space for binary SSA, Eq. 17 is used.

$$\text{Dissimilarity}(X_{\text{new}}, X_i) \approx \varphi \times \text{Similarity}(X_i, Y_i) \tag{17}$$

where X_{new} is the newly generated spider (the candidate solution) for the current spider (X_i), X_i is the current spider, Y_i is a random spider and φ is a positive random scaling factor. And \approx is the almost equal operator. In other words, to produce the new binary solution X_{new} , the value of the following three variables (M_{11} , M_{10} and M_{01}) must be determined.

$$\min \left| \left(1 - \frac{M_{11}}{M_{11} + M_{10} + M_{01}} \right) - \varphi \times \left(1 - \frac{m_{11}}{m_{11} + m_{10} + m_{01}} \right) \right| \tag{18}$$

$$M_{11} + M_{01} = n1 \tag{19}$$

$$M_{10} \leq n0 \tag{20}$$

$$M_{10}, M_{01}, M_{11} \geq 0 \text{ and integer} \tag{21}$$

where $n1$ is the total number of bits with value 1 in X_i and $n0$ is the total number of bits with value 0 in X_i . For determining the values of M_{01} , M_{10} and M_{11} , the integer mathematical model given by Eqs. (18)–(21) must be solved. After solving the mathematical model and getting the optimal values of M_{01} , M_{10} and M_{11} , the X_{new} candidate solution can be obtained using the new binary solution generator (NBSG). The workings of the NBSG algorithm are explained with the example for 16-bit individuals below:

Let $X_i = [0011010110101110]$, $Y_i = [1000101111000110]$ and $\varphi = 0.7$

$m_{11} = 4$, $m_{01} = 4$, $m_{10} = 5$, $m_{00} = 3$ [m_{00} is not used in Jaccard’s similarity measurement technique (Ling et al. 2010b)].

$$\begin{aligned} \text{Dissimilarity}(X_i, Y_i) &= 1 - \text{Similarity}(X_i, Y_i) \\ &= 1 - \frac{m_{11}}{m_{01} + m_{10} + m_{11}} = 1 - \frac{4}{4 + 4 + 5} = 1 - \frac{4}{13} = 0.69 \\ A &= \varphi \times \text{Dissimilarity}(X_i, Y_i) = 0.7 \times 0.69 = 0.48 \end{aligned}$$

According to the integer mathematical model given by Eqs. (18)–(21):

$$\begin{aligned} \min f & \left| \left(1 - \frac{M_{11}}{M_{11} + M_{10} + M_{01}} \right) - A \right| \\ &= \min f \left| \left(1 - \frac{M_{11}}{M_{11} + M_{10} + M_{01}} \right) - 0.48 \right| \end{aligned}$$

$$M_{11} + M_{01} = 8(n1)$$

$$M_{10} \leq 8(n0)$$

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

After the model is solved by using an enumeration scheme, the optimal output is obtained as $M_{01} = 0$, $M_{10} = 5$ and $M_{11} = 8$ and $f = 0.1$. After all, X_{new} is generated as a 1×16 zero-bit array by using the NBSG algorithm given as follows:

Inheritance phase The positions of the bits that are equal to 1 in X_i are found ($Ones_{X_i} = [3, 4, 6, 8, 9, 11, 13, 14, 15]$) and $M_{11} = 8$ bit positions are selected from the $Ones_{X_i}$ (assume that the random selection $[3, 4, 6, 8, 9, 11, 13, 15]$, and the bits of X_{new} in these positions are changed to 1. The new state of X_{new} is $[00111010110101010]$.

Disinheritance phase The positions of the bits that are equal to 0 in X_i are found ($Zeros_{X_i} = [1, 2, 5, 7, 10, 12, 16]$) and the $M_{10} = 5$ bit position is selected from the $Zeros_{X_i}$ (assume that the random selection includes $[1, 2, 5, 12, 16]$), and the bit of X_{new} in these positions is changed to 1. After this change, the final state of the candidate solution X_{new} is $[1111110110111011]$.

Some examples of using the NBSG algorithm can be seen in (Çinar and Kiran 2018; Jaccard 1901; Choi et al. 2010). NBSG algorithm operation steps are shown in Table 3.

3.3 Logic gate-based social spider algorithm (LogicSSA)

Logic gates are very suitable for working on binary structures in binary solution space. The input and output values of these gates are binary values $\{0, 1\}$. There are basically three gates. These are “AND,” “OR” and “Exclusive or (XOR)” logic gates. *Exclusive or (XOR)* logic gate is a combination of “AND” and “OR” logic gates and is widely used in logical situations (Çinar and Kiran 2018; Aslan et al. 2019; Kiran and Gunduz 2013). In LogicSSA, the “XOR” logic gate is used in a new candidate solution (new spider) production stage. Because the changing probability of the bit (bit value = 0 and bit value = 1) in the solution is 50% in this gate. In the “OR” logic gate, the changing probability of the bit (bit value = 1) in the solution is 75% and the changing probability of the bit (bit value = 0) in the solution is 25. In the “AND” logic gate,

the changing probability of the bit (bit value = 1) in the solution is 25% and the changing probability of the bit (bit value = 0) in the solution is 75. In the “XOR” logic gate, the probabilities are equal to each other. The truth tables and these situations are given in Table 4. “+” symbol represents “OR” logic gate, “&” symbol represents “AND” logic gate and “⊕” symbol represents “XOR” logic gate. In order to improve the search capability of binary SSA, the XOR gate is used in candidate solution production and the behavior of the ST control parameter is slightly changed. In this paper, the control parameter value of ST is taken as 0.3 (Çinar and Kiran 2018). In LogicSSA, a new candidate solution (spider) has been created with Eq. 22.

$$S_{kj} = \begin{cases} X_{ij} \oplus (B_j \oplus N_{rj}), & \text{if } (\text{rand}_{ij} < \text{ST}) \\ X_{ij}, & \text{otherwise} \end{cases} \quad (22)$$

where S_{kj} is the j th dimension of k th new spider (candidate solution) produced for the i th spider, X_{ij} is the j th dimension of i th spider, B_j is the j th dimension of best spider obtained so far and N_{rj} is the j th dimension of neighbor spider randomly selected from the search space.

3.4 Binary SSA based on similarity measurement and logic gate

Binary SSA is an algorithm consisting of a combination of LogicSSA and SimSSA techniques. We use both techniques in the production of candidate solutions in binary SSA. One of the SimSSA or LogicSSA techniques is selected according to a randomly generated variable in the $[0, 1]$ range. This selection process is shown in Eq. 23.

Candidate Solution Production Method

$$= \begin{cases} \text{SimSSA}, & \text{if } (\text{rand} < 0.5) \\ \text{LogicSSA}, & \text{otherwise} \end{cases} \quad (23)$$

In the binary SSA approach, one of the SimSSA or LogicSSA techniques is selected for the production of new candidate solutions. Both techniques (SimSSA and LogicSSA) in search space have some advantages. LogicSSA has a strong local search capacity, and SimSSA is good at discovering the points on the solution space. Therefore,

Table 3 NBSG algorithm (Jaccard 1901)

Step 1. Compute the value of A through φ .Dissimilarity(X_i, Y_i) and use it in the mathematical programming model (18)–(21) with output M_{01} , M_{10} and M_{11} . Apply the total enumeration (TE) scheme to solve the mathematical programming problem optimally. Initialize by a $1 \times D$ zero solution vector

Step 2. (*Inheritance phase*). Based on any logic, select M_{11} number of zero bits from X_{new} which their corresponding value in X_i is 1. Change the value of the selected bits from 0 to 1

Step 3. (*Disinheritance phase*). Based on any logic, select M_{10} number of zero bits from which their corresponding value in X_i is 0. Change the value of the selected bits from 0 to 1. Then, report the new binary solution vector X_{new} as output

Table 4 OR, AND and XOR truth tables

| Positions | | OR gate | AND gate | XOR gate |
|-----------|-------|------------------|------------------|------------------|
| X_i | X_k | $X_i + X_k$ | $X_i \& X_k$ | $X_i \oplus X_k$ |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| | | 75% = 1, 25% = 0 | 25% = 1, 75% = 0 | 50% = 1, 50% = 0 |

binary SSA based on SimSSA and LogicSSA is also useful to provide a balanced search in the solution space. So new candidate solutions (new spiders) are produced by using similarity measurement and logic gate techniques, and more efficient individuals are obtained by comparing with the current individuals. Since both techniques have advantages in different views, the probability of selecting any of the techniques is determined as 0.5. This fixed selection rate can be changed according to the structure of BOPs. The detailed pseudocode of the binary SSA is shown in Fig. 4.

According to Fig. 4, the proposed binary SSA consists of three main stages. In the first stage, the basic parameters are defined and random binary positions for the social spiders are determined in binary space. In the second stage, the fitness function is run in binary space for the spiders and the best fitness value is calculated. Vibrations are calculated for each spider, random walking is performed. Spider positions in the continuous search space are calculated according to the transfer function, and their equivalents in the binary space are determined. They are converted to binary $\{0, 1\}$ according to a certain threshold value. In addition, we show the binary SSA as binary SSA-Tanh when $k = 1$, binary SSA-Sigm when $k = 2$, binary SSA-MSigm when $k = 3$ and binary SSA-Arctan when $k = 4$. In the third stage, the candidate solutions production method (including SimSSA and LogicSSA) is run to produce candidate solutions (new spiders). Produced candidate solutions are compared with the existing spider population. If the fitness values of the candidate solutions are better than the fitness values of the current population individuals, the population of individuals is swapped mutually with the candidate solutions. Otherwise, no changes will be made. The best result is obtained.

In this paper, the continuous test function is handled as a BOP where each dimension is encoded by Q -bit binary string such that first bit for the sign of the number, the subsequent P bits for the integer part and the final $Q - P - 1$ bits for a fractional part of the number. The values of Q and P were chosen for each test function according to the range values of the number. For example, for $f1$ benchmark function, we used $Q = 13$, $P = 7$, the first bit of $Q =$ the sign of bit and a fractional part = $Q - (P + 1)$ (the

sign bit) = $13 - (7 + 1) = 5$ where the problem is handled with binary strings until the termination condition is satisfied. Afterward, each of these Q -bit strings is converted into a real number, and the fitness value is computed.

4 Experimental results and analysis

Binary social spider algorithm is tested on MATLAB R2014a that installed over windows 7, 64 bit, the system of 2.30 GHz processor with 4 GB RAM. Each binary SSA variation is tested twenty independent runs with the same maximum number of iterations for each run. To have fair comparisons, all binary SSA variations have been carefully run in the same programming language, similar platform and similar parameter values. In experimental studies, eighteen different unimodal and multimodal benchmark functions, which are shown in Table 5, are solved separately. Wilcoxon signed-rank test is applied for the variations of binary SSA for obtained fitness values of spider populations.

Benchmark functions divided into two groups according to their type. These functions are obtained from various sources (Yu and Li 2015; Rizk-Allah et al. 2018; Surjanovic and Bingham 2019; Jamil and Yang 2013).

Group I Unimodal benchmark functions $f1$ – $f11$, which has only one global optimum and can evaluate the exploitation capability of the investigated algorithms.

Group II Functions $f12$ – $f18$ (multimodal), which include many local optimal whose number increases exponentially with the problem size, become highly useful when the purpose is to evaluate the exploration capability of the investigated algorithms.

For each benchmark function, four measures are used to evaluate the performance of each algorithm to solve that function. These measures include (1) average (mean), (2) standard deviation (sd), (3) minimum value (best) and (4) maximum value (worst) any they defined as in Table 6. n_{run} represents the total number of runs.

Fig. 4 Detailed pseudocode of the binary SSA

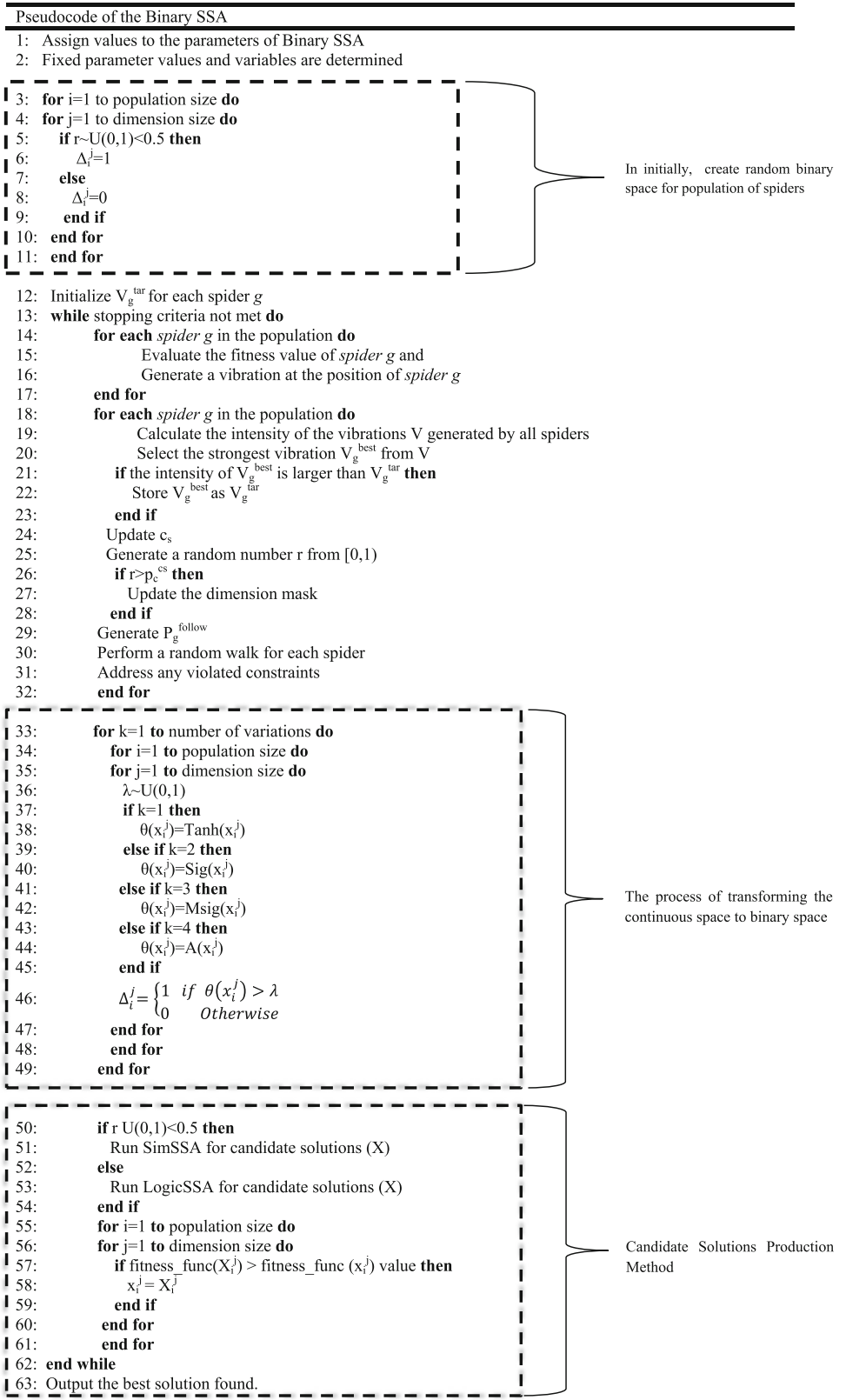


Table 5 Classical benchmark functions used in the experimental study

| Type | Name | Function | Range | f_{minimum} |
|------------|----------|---|-------------------|--|
| Unimodal | f_1 | $f_1(X) = \sum_{i=1}^D x_i^2$ | $[-100, 100]^D$ | $X^* = (0, \dots, 0); f(X^*) = 0$ |
| | f_2 | $f_2(X) = \sum_{i=1}^D ix_i^4$ | $[-2.56, 2.56]^D$ | $X^* = (0, \dots, 0); f(X^*) = 0$ |
| | f_3 | $f_3(X) = \sum_{i=1}^D [(ix_i + 0.5)]^2$ | $[-10, 10]^D$ | $X^* = (0, \dots, 0); f(X^*) = 0$ |
| | f_4 | $f_4(x) = \sum_{i=1}^D ix_i^4 + \text{random}(0, 1)$ | $[-2.56, 2.56]^D$ | $X^* = (0, \dots, 0); f(X^*) = 0$ |
| | f_5 | $f_5(x) = \max_i \{ x_i , 1 \leq i \leq D\}$ | $[-100, 100]^D$ | $X^* = (0, \dots, 0); f(X^*) = 0$ |
| | f_6 | $f_6(X) = \sum_{i=1}^D x_i + \prod_{i=1}^D x_i $ | $[-100, 100]^D$ | $X^* = (0, \dots, 0); f(X^*) = 0$ |
| | f_7 | $f_7(X) = \sum_{i=1}^{D-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$ | $[-30, 30]^D$ | $X^* = (0, \dots, 0); f(X^*) = 0$ |
| | f_8 | $f_8(X) = \sum_{i=1}^D ix_i^2$ | $[-10, 10]^D$ | $X^* = (0, \dots, 0); f(X^*) = 0$ |
| | f_9 | $f_9(X) = (x_1 - 1)^2 + \sum_{i=1}^D i(2x_i^2 - x_{i-1})^2$ | $[-10, 10]^D$ | $X^* = (0, \dots, 0); f(X^*) = 0$ |
| | f_{10} | $f_{10}(X) = \sum_{i=1}^D x_i ^{1+1}$ | $[-1, 1]^D$ | $X^* = (0, \dots, 0); f(X^*) = 0$ |
| | f_{11} | $f_{11}(X) = \sum_{i=1}^D \sum_{j=1}^D x_i^2 x_j^2$ | $[-65, 65]^D$ | $X^* = (0, \dots, 0); f(X^*) = 0$ |
| Multimodal | f_{12} | $f_{12}(x) = 1 - \cos\left(2\pi\sqrt{\sum_{i=1}^D x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^D x_i^2}$ | $[-100, 100]^D$ | $X^* = (0, \dots, 0); f(X^*) = 0$ |
| | f_{13} | $f_{13}(X) = \sum_{i=1}^{10} (x_i - 1)^2 - \sum_{i=2}^{10} x_i x_{i-1}$ | $[-n^2, n]^D$ | $i = 1, 2, \dots, D; x_i = (D + 1 - i)f(X^*) = -D(D + 4)(D - 1)/6$ |
| | f_{14} | $f_{14}(X) = 0.1 \left\{ \sin^2(3\pi x_1) + \sum_{i=1}^D (x_i - 1)^2 [1 + \sin^2(3\pi x_i + 1)] \right\} + (x_D - 1)^2 [1 + \sin^2(2\pi x_D)]$ | $[-10, 10]^D$ | $X^* = (1, \dots, 1); f(X^*) = 0$ |
| | | $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i - a)^m & x_i < -a \end{cases}$ | | |
| | f_{15} | $f_{15}(X) = \sum_{i=1}^D [x_i^2 - 10 \cos(2\pi x_i) + 10]$ | $[-50, 50]^D$ | $X^* = (0, \dots, 0); f(X^*) = 0$ |
| | f_{16} | $f_{16}(X) = \frac{1}{4000} \sum_{i=1}^D x_i^2 - \prod_{i=1}^D \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$ | $[-600, 600]^D$ | $X^* = (0, \dots, 0); f(X^*) = 0$ |
| | f_{17} | $f_{17}(X) = -20 \exp\left(-0.2\sqrt{\frac{1}{D} \sum_{i=1}^D x_i^2}\right) - \exp\left(\frac{1}{D} \sum_{i=1}^D \cos(2\pi x_i)\right) + 20 + \exp$ | $[-32, 32]^D$ | $X^* = (0, \dots, 0); f(X^*) = 0$ |
| | f_{18} | $f_{18}(X) = \sum_{i=1}^D -x_i \sin(\sqrt{ x_i })$ | $[-500, 500]^D$ | $X^* = (420, \dots, 420); f(X^*) = -418.9829 \times D$ |

Table 6 Performance measure

| | |
|--------------------|---|
| Mean | $Mean = \frac{1}{n_{run}} \times \sum_{i=1}^{n_{run}} Fitness_func_i$ |
| Standard deviation | $SD = \sqrt{\frac{1}{n_{run}} \sum_{i=1}^{n_{run}} (gbest_i - Mean)^2}$ |
| Best | $Fitness_func_Best = \min_{i=1:n_{run}} Fitness_func_Best_i$ |
| Worst | $Fitness_func_Worst = \max_{i=1:n_{run}} Fitness_func_Worst_i$ |

4.1 Determination of initial population size on the fitness function

Different combinations of population size are tested on $f1$, $f5$, $f15$ and $f17$ selected randomly unimodal and multimodal benchmark functions. The results are shown in Table 7. According to the results, the most appropriate values are determined as 30 for the initial population size.

The same parameters used in the variations of binary SSA comparison, which are population size (N), maximum iteration value and other parameters, are shown in Table 8.

4.2 Comparison between different variations of binary SSA

The variations of binary SSA based on SimSSA and LogicSSA (i.e., binary SSA-Tanh, binary SSA-Sigm, binary SSA-MSigm and binary SSA-Arctan) are tested in eighteen different unimodal and multimodal benchmark functions. Each benchmark function is run 20 times. The best, worst,

mean and standard deviation (SD) are done for obtained benchmark function results. The best, worst, mean and standard deviation (SD) comparison results, which are obtained for unimodal benchmark functions ($f1$ – $f11$) by selecting population size (N) = 30 and maximum iteration = 50 values for binary SSA, are shown in Table 9. The standard deviation results, the mean results and the best results of variations of binary SSA’s superior performance are marked in bold.

According to the standard deviation results, binary SSA-Arctan has shown better performance than binary SSA-Tanh, binary SSA-Sigm and binary SSA-MSigm in all unimodal benchmark functions. The binary SSA-Arctan variation shows success at roughly 81.81% of unimodal benchmark functions for 9 out of 11 benchmark functions ($f1$, $f2$, $f3$, $f4$, $f5$, $f6$, $f7$, $f9$ and $f10$). According to the mean results, binary SSA-Arctan has shown better performance than binary SSA-Tanh, binary SSA-Sigm and binary SSA-MSigm in all unimodal benchmark functions. The binary SSA-Arctan variation shows success at roughly 63.63%

Table 7 Mean of the fitness values according to different combinations of population size (N) on $f1$, $f5$, $f15$ and $f17$

| Variations of binary SSA | Population size (N) | $f1$ | $f5$ | $f15$ | $f17$ |
|--------------------------|-------------------------|------------------|---------------|----------------|----------------|
| Binary SSA-Tanh | 10 | 0.0000 | 2.0000 | 34.0000 | 25.0000 |
| | 20 | 0.0000 | 0.0000 | 32.0000 | 22.0000 |
| | 30 | 0.0000 | 0.0000 | 0.0000 | 20.0000 |
| Binary SSA-Sigm | 10 | 12.5000 | 11.5000 | 20.0000 | 21.0000 |
| | 20 | – 53.0000 | 11.4000 | 6.0000 | 21.0000 |
| | 30 | – 23.0000 | 9.0000 | 14.0000 | 20.0000 |
| Binary SSA-MSigm | 10 | 80.0000 | 9.0000 | 50.0000 | 20.0000 |
| | 20 | 1.7000 | 7.5000 | 32.0000 | 20.0000 |
| | 30 | 0.0000 | 2.6000 | 21.0000 | 20.0000 |
| Binary SSA-Arctan | 10 | 0.0000 | 10.5000 | 45.0000 | 1.2E–04 |
| | 20 | 0.0000 | 0.0000 | 0.0000 | 1.2E–05 |
| | 30 | 0.0000 | 0.0000 | 0.0000 | 1.2E–06 |

The best results are marked with bold

Table 8 Parameter setup for binary SSA variations (i.e., binary SSA-Tanh, binary SSA-Sigm, binary SSA-MSigm and binary SSA-Arctan)

| Parameters | Values |
|-------------------------|---|
| Population size (N) | 30 |
| Maximum iteration | 50 |
| Number of runs | 20 |
| Other parameters | $r_a = 1, p_c = 0.7; p_m = 0.1; ST = 0.3$ (Yu and Li 2015; Emary et al. 2016) |

Table 9 Comparison of binary SSA-Tanh, binary SSA-Sigm, binary SSA-MSigm and binary SSA-Arctan with population size = 30 in unimodal benchmark functions with the maximum number of iterations = 50

| Function | Binary SSA-Tanh | Binary SSA-Sigm | Binary SSA-MSigm | Binary SSA-Arctan |
|------------|-----------------|------------------|------------------|-------------------|
| <i>f1</i> | | | | |
| Best | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Mean | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Worst | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| SD | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| <i>f2</i> | | | | |
| Best | 0.0000 | 2.0000 | 2.0000 | 0.0000 |
| Mean | 1.0124 | 2.0410 | 2.1045 | 1.0214 |
| Worst | 1.5600 | 2.5600 | 2.3600 | 1.1500 |
| SD | 0.0280 | 0.2800 | 0.4500 | 0.0250 |
| <i>f3</i> | | | | |
| Best | 0.0000 | 6.0001 | 7.5000 | 0.0000 |
| Mean | 0.0000 | 6.0001 | 7.5000 | 0.0000 |
| Worst | 0.0000 | 6.0001 | 7.5000 | 0.0000 |
| SD | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| <i>f4</i> | | | | |
| Best | 1.26E-05 | 4.26E-04 | 4.25E-04 | 1.22E-05 |
| Mean | 0.000135 | 0.0535 | 0.0452 | 1.85E-04 |
| Worst | 0.00157 | 7.56E-03 | 6.85E-03 | 2.77E-03 |
| SD | 0.0006 | 0.0007 | 0.0007 | 0.0006 |
| <i>f5</i> | | | | |
| Best | 0.0000 | 68.0050 | 55.0400 | 0.0000 |
| Mean | 1.0100 | 75.0900 | 60.7580 | 0.0000 |
| Worst | 6.0000 | 85.0600 | 65.7700 | 0.0000 |
| SD | 2.1200 | 10.0080 | 3.0020 | 0.0000 |
| <i>f6</i> | | | | |
| Best | 0.0000 | 62.4250 | 56.7450 | 0.0000 |
| Mean | 0.0000 | 63.2500 | 66.9800 | 0.0000 |
| Worst | 0.0000 | 68.1100 | 87.5500 | 0.0000 |
| SD | 0.0000 | 5.4780 | 11.2540 | 0.0000 |
| <i>f7</i> | | | | |
| Best | 0.0000 | 4.0001 | 8.012700 | 0.0000 |
| Mean | 13.0017 | 17.0127 | 15.1027 | 8.4150 |
| Worst | 18.0020 | 35.0002 | 29.0001 | 14.0000 |
| SD | 0.0129 | 0.1441 | 0.02100 | 0.0128 |
| <i>f8</i> | | | | |
| Best | 0.0000 | - 15.0001 | - 15.0001 | - 15.0001 |
| Mean | 4.0174 | - 12.0014 | - 10.0042 | - 6.4710 |
| Worst | 10.0000 | 15.0001 | 15.0001 | 12.0002 |
| SD | 0.3340 | 0.3210 | 0.3220 | 0.3240 |
| <i>f9</i> | | | | |
| Best | 0.0000 | 1.00E-05 | 0.0000 | 0.0000 |
| Mean | 2.1700 | 7.1700 | 0.0000 | 0.0000 |
| Worst | 10.0000 | 10.0000 | 0.0000 | 0.0000 |
| SD | 3.6704 | 4.8448 | 0.0000 | 0.0000 |
| <i>f10</i> | | | | |
| Best | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Mean | 0.0000 | 5.00E-01 | 0.0000 | 0.0000 |

Table 9 (continued)

| Function | Binary SSA-Tanh | Binary SSA-Sigm | Binary SSA-MSigm | Binary SSA-Arctan |
|------------|-----------------|-----------------|------------------|-------------------|
| Worst | 0.0000 | 1.00E+00 | 0.0000 | 0.0000 |
| SD | 0.0000 | 0.5000 | 0.0000 | 0.0000 |
| <i>f11</i> | | | | |
| Best | 0.0000 | 20.0000 | 0.0000 | 0.0000 |
| Mean | 1.3300 | 35.2000 | 1.00E-05 | 2.50E-05 |
| Worst | 8.0001 | 65.0000 | 4.00E-05 | 9.00E-05 |
| SD | 2.9800 | 13.0000 | 1.53E-05 | 3.25E-05 |

SD is the standard deviation; best is the minimum value; worst is the maximum value; the mean is the average of the unimodal benchmark function results

unimodal benchmark functions for 7 out of 11 benchmark functions (*f1*, *f3*, *f5*, *f6*, *f7*, *f9* and *f10*). According to the best results, binary SSA-Arctan has shown high performance in all unimodal benchmark functions. The binary SSA-Arctan variation shows success at roughly 100% of unimodal benchmark functions for 11 out of 11 benchmark functions. The reason for this success is due to a more balanced diversification between the candidate solutions with the similarity measurement and logic gate techniques and type of the transfer function.

The best, worst, mean and standard deviation (SD) comparison results, which are obtained for multimodal benchmark functions (*f11*–*f18*) by selecting population size (N) = 30 and maximum iteration = 50 values for binary SSA, are shown in Table 10.

For Table 10, according to the standard deviation results, binary SSA-Arctan has shown better performance than binary SSA-MSigm, binary SSA-Sigm and binary SSA-Tanh in all multimodal benchmark functions. The binary SSA-Arctan variation shows success at roughly 71.43% of multimodal benchmark functions for 5 out of 7 benchmark functions (*f12*, *f13*, *f14*, *f15* and *f18*). According to the mean results, binary SSA-Arctan has shown better performance than binary SSA-MSigm, binary SSA-Sigm and binary SSA-Tanh in all multimodal benchmark functions. The binary SSA-Arctan variation shows success at roughly 57.14% of multimodal benchmark functions for 4 out of 7 benchmark functions (*f13*, *f14*, *f15* and *f18*). According to the best results, binary SSA-Arctan has shown better performance than binary SSA-Tanh, binary SSA-Sigm and binary SSA-MSigm in all multimodal benchmark functions. The binary SSA-Arctan variation shows success at roughly 57.14% of multimodal benchmark functions for 4 out of 7 benchmark functions (*f13*, *f14*, *f15* and *f16*). According to the best results, binary SSA-Arctan and binary SSA-MSigm variations show success at roughly 57.14% of multimodal benchmark functions for 4 out of 7 benchmark functions and binary SSA-Tanh variation shows success at roughly 71.43% of multimodal benchmark functions for 5 out of 7 benchmark

functions. According to the results, binary SSA-Tanh and binary SSA-Arctan have shown superior performance compared to other variations of binary SSA according to many measurement criteria (best, worst, mean and SD). Based on the observed results, it can conclude that binary SSA-Tanh and binary SSA-Arctan have good quality and stability than the others. Binary SSA-Tanh and binary SSA-Arctan variations have established a more stable balance between exploration and exploitation.

The comparison results of fitness value, which are obtained for eighteen unimodal and multimodal benchmark functions by selecting population (N) size as 30 and maximum iteration as 50 for variations of binary SSA, are shown in Fig. 5. Based on the convergence graphics of the benchmark functions, three significant conclusions can be drawn: (1) Binary SSA-Tanh and binary SSA-Arctan provide the best convergence speed on almost selected functions, and also, binary SSA-Tanh and binary SSA-Arctan perform better than the other comparative variations of binary SSA. (2) The binary SSA-Tanh and binary SSA-Arctan provide stable solutions on almost selected benchmark functions, which imply the reliability of solutions, especially for large-scale instances. (3) Most results of the binary SSA-Tanh, binary SSA-MSigm and binary SSA-Arctan exhibit a continuous improvement during the optimization process.

In particular, the variation of binary SSA-Arctan has quickly reached an optimum solution in almost all benchmark test functions. The convergence speed of this algorithm is increased by the logic gate and similarity measurement techniques added during the candidate solution production stage. With the similarity measurement technique (SimSSA), this algorithm has succeeded in finding new points in binary search space and finding global optimum instead of local optimums. Logic gates, a new and successful technique, have become very popular in recent years. In this paper, the LogicSSA technique is proposed, which is based on logic gates. With the LogicSSA technique, binary SSA has been saved from local

Table 10 Comparison of binary SSA-Tanh, binary SSA-Sigm, binary SSA-MSigm and binary SSA-Arctan with population size = 30 in multimodal benchmark functions with the maximum number of iterations = {50, 50, 50, 50}

| Function | Binary SSA-Tanh | Binary SSA-Sigm | Binary SSA-MSigm | Binary SSA-Arctan |
|------------|-------------------|-------------------|-------------------|-------------------|
| <i>f12</i> | | | | |
| Best | 4.0000 | 3.0000 | 1.000 | 17.0001 |
| Mean | 35.7000 | 64.800 | 40.3000 | 54.0000 |
| Worst | 80.0010 | 100.0000 | 91.0001 | 86.0001 |
| SD | 24.0040 | 30.0745 | 29.7247 | 22.2411 |
| <i>f13</i> | | | | |
| Best | - 127.0000 | - 127.0000 | - 127.0000 | - 127.0000 |
| Mean | - 19.500 | - 57.5000 | - 124.0000 | - 127.0000 |
| Worst | 127.0000 | - 125.0000 | - 111.0000 | - 127.0000 |
| SD | 67.8939 | 85.9569 | 5.9628 | 0.0000 |
| <i>f14</i> | | | | |
| Best | 0.0000 | - 15.0001 | - 15.0002 | - 15.0002 |
| Mean | 2.8000 | - 0.333 | - 15.0001 | - 15.0002 |
| Worst | 6.0000 | 14.0000 | - 15.0001 | - 15.0002 |
| SD | 2.3152 | 10.8884 | 3.73E-05 | 1.78E-15 |
| <i>f15</i> | | | | |
| Best | 0.0000 | 31.0540 | 0.0000 | 0.0000 |
| Mean | 25.0050 | 41.0010 | 29.0410 | 0.0000 |
| Worst | 47.0140 | 49.0140 | 38.0740 | 0.0000 |
| SD | 75.1000 | 55.1000 | 95.1000 | 0.0000 |
| <i>f16</i> | | | | |
| Best | 0.0000 | 529.4650 | 365.7485 | 0.0000 |
| Mean | 0.0000 | 580.4650 | 368.7480 | 0.25470 |
| Worst | 0.0000 | 600.0000 | 524.7840 | 0.43257 |
| SD | 0.0000 | 233.4580 | 324.5740 | 0.09684 |
| <i>f17</i> | | | | |
| Best | 20.0015 | 20.5471 | 20.4175 | 20.5472 |
| Mean | 20.0478 | 20.9870 | 21.1004 | 20.5147 |
| Worst | 20.5410 | 21.0141 | 21.9840 | 20.5470 |
| SD | 0.0147 | 0.0502 | 1.1470 | 0.0287 |
| <i>f18</i> | | | | |
| Best | - 500.0000 | - 421.1470 | - 435.8540 | - 498.0178 |
| Mean | - 468.1247 | - 395.1240 | - 400.2140 | - 489.6045 |
| Worst | - 250.7480 | - 120.0124 | - 80.1450 | - 330.5480 |
| SD | 250.4012 | 240.1204 | 410.2100 | 140.2140 |

The best results are marked with bold

SD is the standard deviation; best is the minimum value; worst is the maximum value; mean is the average of the multimodal benchmark function results

traps in the local search space and the ability to discover local optimums has been increased.

4.3 Wilcoxon signed-rank test and evaluation of binary SSA variations

In this paper, variations of binary SSA are operated with 20 times various eighteen unimodal and multimodal benchmark functions with population size = 30 and maximum iteration = 50. Twenty trials are done in each of the

unimodal and multimodal benchmark functions, and twenty data sets are obtained. Whether there is a significant difference in this data set is searched by using the Wilcoxon signed-rank test (Derrac et al. 2011). The test results are shown in Table 11 for unimodal benchmark functions, and Table 12 shows multimodal benchmark functions. In Wilcoxon signed-rank test, alpha = 0.05 meaning level and determined value are 52 for $n = 10$ (n is data set number) (Acilar 2013). Two hypotheses are evaluated in the results. In the H_0 hypothesis, $M_{data1} = M_{data2}$, and in H_1 hypothesis,

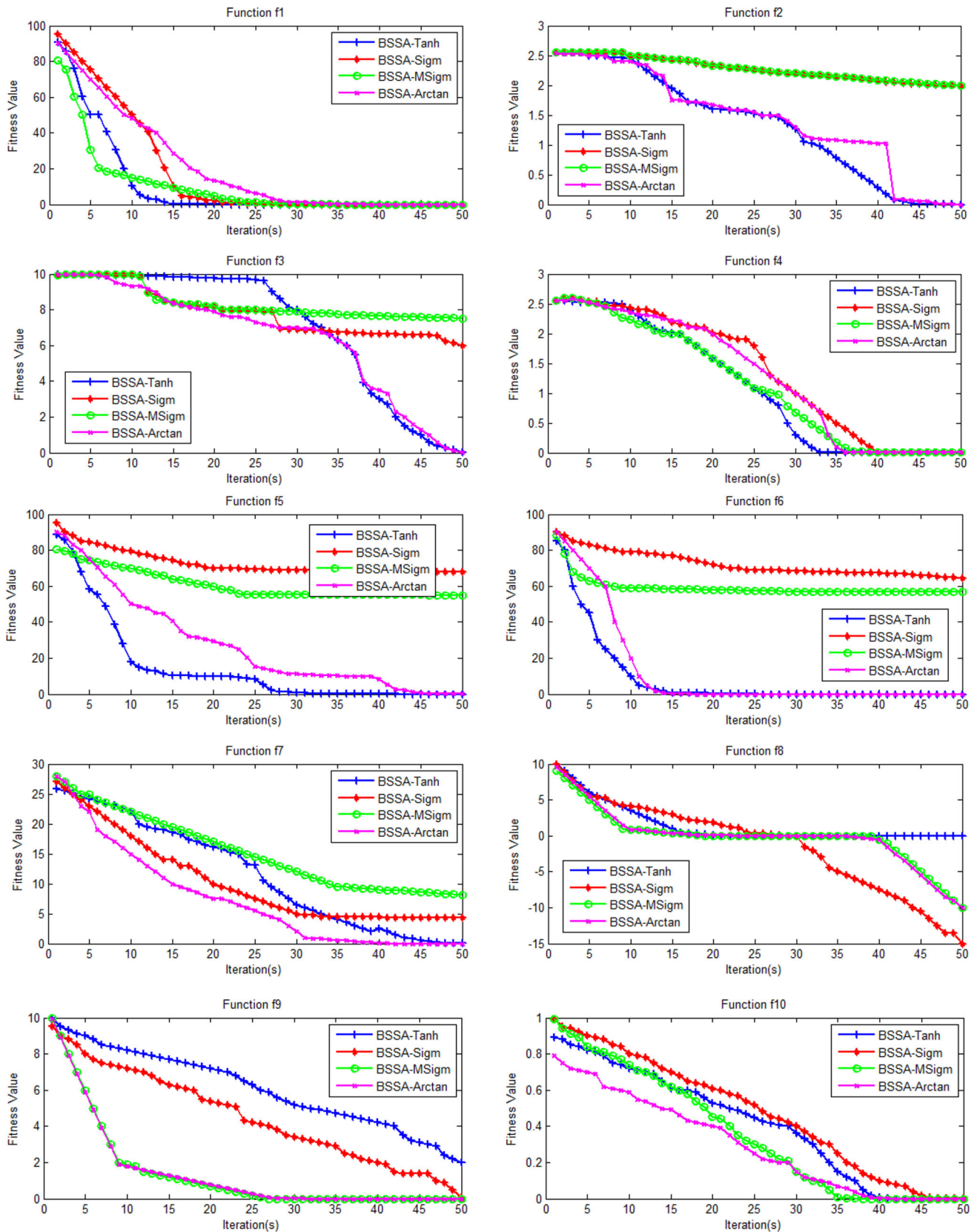


Fig. 5 Convergence graphics for variations of binary SSA for benchmark functions (f1–f18)

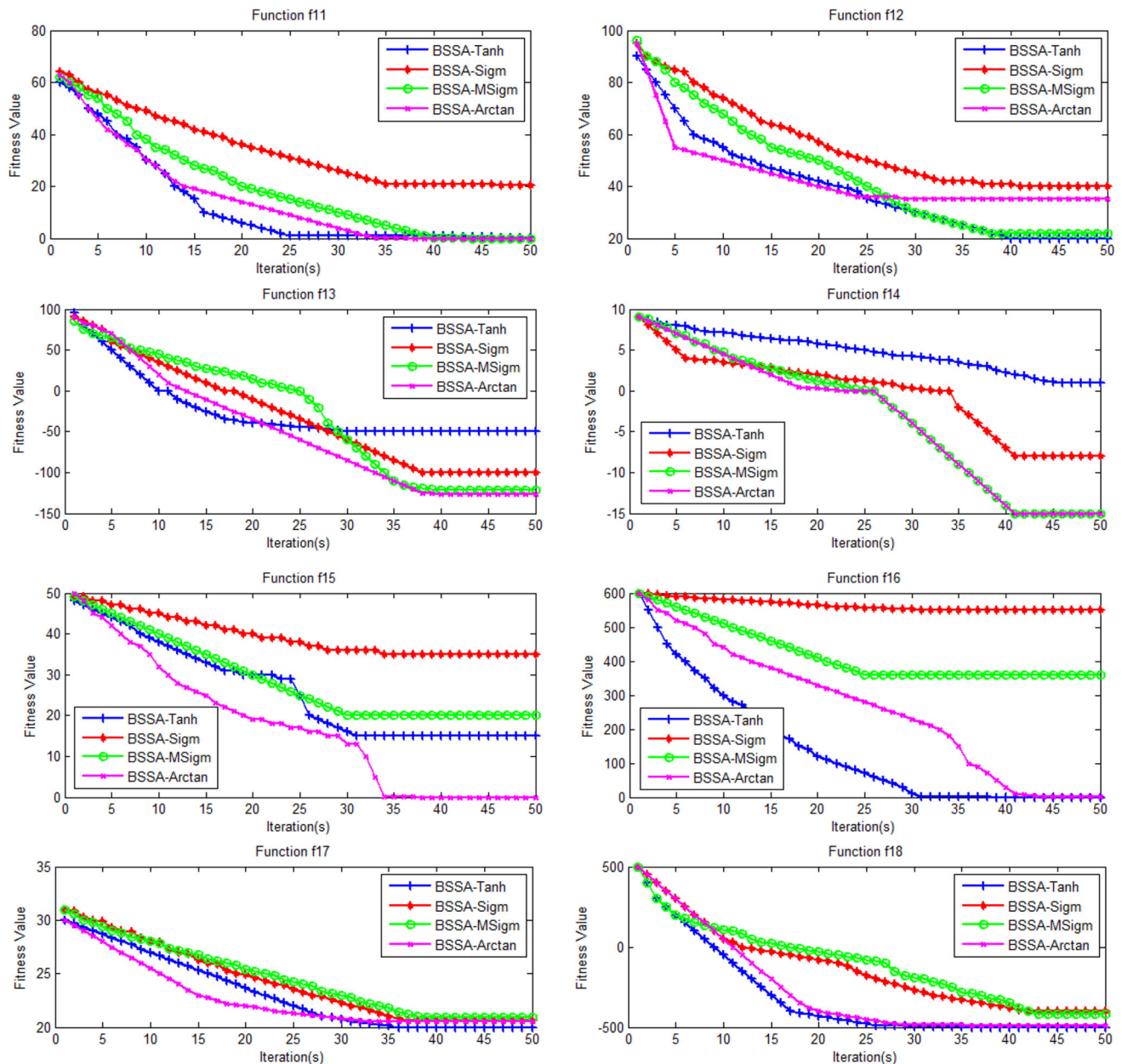


Fig. 5 continued

$M_{data1} \neq M_{data2}$. H_0 hypothesis is rejected in $h = 1$ values and it is called that there is a semantic difference between the algorithms. The p values of the hypothesis are calculated by using MATLAB R2014a software. By these hypothesizes, the superiority among two optimizers can be determined. In this paper, we investigated whether there is a significant difference between variations of the binary SSA results. All variations of binary SSA are tested with Wilcoxon signed-rank test. If the obtained results in benchmark functions are equal, the Wilcoxon signed-rank test is not applied and the result is “NaN.” According to

the obtained results through implementing the Wilcoxon signed-rank test for pair-wise comparisons between variations of the binary SSA, there is not a semantic difference between the results obtained from binary SSA-Arctan and binary SSA-MSigm.

Table 13 shows the obtained results through implementing the Wilcoxon signed-rank test for comparisons between binary SSA-Arctan and the other variations of the binary SSA. From these results, we can conclude that the binary SSA-Arctan has a superior performance over the other variations.

Table 11 Wilcoxon signed-rank test on variations of binary SSA in the maximum number of iterations = 50 and population size = 30 in unimodal benchmark functions ($f1-f11$)

| <i>F</i> | (Binary SSA-Arctan) – (Binary SSA-Sigm) | | (Binary SSA-Arctan) – (Binary SSA-MSigm) | | (Binary SSA-Arctan) – (Binary SSA-Tanh) | | (Binary SSA-Sigm) – (Binary SSA-MSigm) | | (Binary SSA-Sigm) – (Binary SSA-Tanh) | | (Binary SSA-MSigm) – (Binary SSA-Tanh) | |
|----------|---|--------------|--|--------------|---|--------------|--|--------------|---------------------------------------|--------------|--|--------------|
| | (<i>P</i>) | (<i>h</i>) | (<i>P</i>) | (<i>h</i>) | (<i>P</i>) | (<i>h</i>) | (<i>P</i>) | (<i>h</i>) | (<i>P</i>) | (<i>h</i>) | (<i>P</i>) | (<i>h</i>) |
| $f1(x)$ | 2.016E–05 | 1 | NaN | 0 | 0.0198 | 1 | 2.02E–05 | 1 | 0.0042 | 1 | 0.0198 | 1 |
| $f2(x)$ | 0.3039 | 0 | 0.6358 | 0 | 0.1403 | 0 | 0.8816 | 0 | 0.4884 | 0 | 0.3363 | 0 |
| $f3(x)$ | 0.0021 | 1 | NaN | 0 | 0.0400 | 1 | 0.0021 | 1 | 0.1607 | 0 | 0.0400 | 1 |
| $f4(x)$ | 0.8710 | 0 | 0.1805 | 0 | 0.7149 | 0 | 0.0759 | 0 | 1 | 0 | 0.0616 | 0 |
| $f5(x)$ | 0.0367 | 1 | 0.4879 | 0 | 0.2420 | 0 | 0.2769 | 0 | 0.1758 | 0 | 0.4508 | 0 |
| $f6(x)$ | 3.72E–04 | 1 | NaN | 0 | 0.005E–04 | 1 | 3.71E–04 | 1 | 0.8217 | 0 | 4.05E–04 | 1 |
| $f7(x)$ | 0.1624 | 0 | 0.1624 | 0 | 0.1624 | 0 | 4.68E–10 | 1 | NaN | 0 | 6.69E–10 | 1 |
| $f8(x)$ | 0.3421 | 0 | NaN | 0 | 1 | 0 | 0.3421 | 0 | 0.5940 | 0 | 1 | 0 |
| $f9(x)$ | 0.002 | 1 | NaN | 0 | 0.3421 | 0 | 0.0020 | 1 | 0.0084 | 1 | 0.3421 | 0 |
| $f10(x)$ | NaN | 0 | NaN | 0 | NaN | 0 | NaN | 0 | NaN | 0 | NaN | 0 |
| $f11(x)$ | 1.096E–06 | 1 | NaN | 0 | 1.64E–04 | 1 | 1.09E–06 | 1 | 0.05 | 1 | 1.64E–04 | 1 |

The *p*-values are below 0.05 for the majority of the benchmark functions

Table 12 Wilcoxon signed-rank test on variations of binary SSA in the maximum number of iterations = 50 and population size = 30 in multimodal benchmark functions ($f12-f18$)

| <i>F</i> | (Binary SSA-Arctan) – (Binary SSA-Sigm) | | (Binary SSA-Arctan) – (Binary SSA-MSigm) | | (Binary SSA-Arctan) – (Binary SSA-Tanh) | | (Binary SSA-Sigm) – (Binary SSA-MSigm) | | (Binary SSA-Sigm) – (Binary SSA-Tanh) | | (Binary SSA-MSigm) – (Binary SSA-Tanh) | |
|----------|---|--------------|--|--------------|---|--------------|--|--------------|---------------------------------------|--------------|--|--------------|
| | (<i>P</i>) | (<i>h</i>) | (<i>P</i>) | (<i>h</i>) | (<i>P</i>) | (<i>h</i>) | (<i>P</i>) | (<i>h</i>) | (<i>P</i>) | (<i>h</i>) | (<i>P</i>) | (<i>h</i>) |
| $f12(x)$ | 0.6072 | 0 | 0.9667 | 0 | 0.9667 | 0 | 0.7972 | 0 | 0.8709 | 0 | 0.7763 | 0 |
| $f13(x)$ | 6.59E–05 | 1 | 6.59E–05 | 1 | 6.59E–05 | 1 | 4.68E–10 | 1 | NaN | 0 | 4.68E–10 | 1 |
| $f14(x)$ | 0.3421 | 0 | 0.3421 | 0 | 0.3421 | 0 | 4.68E–10 | 1 | NaN | 0 | 4.68E–10 | 1 |
| $f15(x)$ | 2.51E–05 | 1 | NaN | 0 | 0.0197 | 1 | 2.52E–05 | 1 | 0.0350 | 1 | 0.0197 | 1 |
| $f16(x)$ | 0.3421 | 0 | NaN | 0 | 0.3421 | 0 | 0.3421 | 0 | 1 | 0 | 0.3421 | 0 |
| $f17(x)$ | 1.09E–06 | 1 | NaN | 0 | 4.01E–04 | 1 | 1.09E–06 | 1 | 0.0192 | 1 | 4.0005E–04 | 1 |
| $f18(x)$ | 3.92E–04 | 1 | 0.9031 | 0 | 5.33E–06 | 1 | 1.54E–04 | 1 | 0.0451 | 1 | 9.83E–07 | 1 |

The *p*-values are below 0.05 for the majority of the benchmark functions

Table 13 Wilcoxon signed-rank test for comparison results in Tables 9 and 10

| Compared approaches | | Evaluations | | | |
|---------------------|------------------|-------------|-------|----------------|-------------------|
| The variations | Compared | R^- | R^+ | <i>p</i> value | Winner |
| Binary SSA-Arctan | Binary SSA-Tanh | 72.5 | 37 | 0.2284 | Binary SSA-Arctan |
| Binary SSA-Arctan | Binary SSA-MSigm | 47 | 33 | 0.9569 | Binary SSA-Arctan |
| Binary SSA-Arctan | Binary SSA-Sigm | 106 | 29 | 0.4617 | Binary SSA-Arctan |

4.4 Comparison of binary SSA with other meta-heuristic algorithms

Case Study 1 Well-known heuristic algorithms of the literature and recently developed binary algorithms are shown in Table 14. In this paper, particle swarm optimization (PSO), genetic algorithm (GA), binary hybrid

particle swarm optimization with wavelet mutation (BHPSOWM), BSalpSA, binary particle swarm optimization (BPSO), binary bat algorithm (BBA), BSCO, BSSO and binary SSA are compared in various unimodal and multimodal benchmark functions in equal comparison parameters. The parameters used in all comparison algorithms are shown in Table 15.

Table 14 Keys commonly used for comparative algorithms

| Key | Method name | References |
|------------|---|---|
| PSO | Particle swarm optimization | Jiang et al. (2017) |
| GA | Genetic algorithm | Jiang et al. (2017) |
| BHPSOWM | Binary hybrid particle swarm optimization with wavelet mutation | Jiang et al. (2017) |
| BSalpSA | Binary salp swarm algorithm | Rizk-Allah et al. (2018) |
| BPSO | Binary particle swarm optimization | Rizk-Allah and Hassanien (2018) |
| BBA | Binary bat algorithm | Rizk-Allah and Hassanien (2018) |
| BSCA | Binary sine cosine algorithm | Rizk-Allah (2018) |
| BSSO | Binary social spider optimization | Cuevas et al. (2013), Shukla and Nanda (2018) |
| Binary SSA | Binary social spider algorithm | |

Table 15 Parameters of all algorithms

| Key | Method name |
|-------------------|---|
| BPSO | Inertia weight: $w_{\min} = 0.4$; $w_{\max} = 0.9$; acceleration coefficients: $c_1 = c_2 = 2$ |
| BBA | Minimum and maximum frequencies: $F_{\min} = 0$ and $F_{\max} = 2$; pulse emission rate: $r = 0.5$; loudness: $A = 0.25$; constants: α ; $\gamma = 0.9$ |
| BSCA | $r_1 = 1-0$ (linear decreasing parameter); $r_2 \in [0, 2\pi]$, $r_3 \in [0, 2]$ |
| BSalpSA | $c_1 = 1-0$ (nonlinear decreasing parameter); $c_2 \in [0, 1]$, $c_3 \in [0, 1]$ |
| Binary SSA-Tanh | $r_a = 1$; $p_c = 0.7$; $p_m = 0.1$; ST = 0.3 |
| Binary SSA-Sigm | $r_a = 1$; $p_c = 0.7$; $p_m = 0.1$; ST = 0.3 |
| Binary SSA-MSigm | $r_a = 1$; $p_c = 0.7$; $p_m = 0.1$; ST = 0.3 |
| Binary SSA-Arctan | $r_a = 1$; $p_c = 0.7$; $p_m = 0.1$; ST = 0.3 |
| Common settings | Population size (N) = 30; number of runs = 20; maximum iterations = 50 |

BPSO, BBA, BSCA, BSalpSA, BSSO and variations of binary SSA (i.e., binary SSA-Tanh, binary SSA-Sigm, binary SSA-MSigm and binary SSA-Arctan) algorithms are compared according to four different criteria. There are the mean, the standard deviation (SD), the best and the worst criteria. In the compared algorithms, the population size (N) is determined as 30 and a maximum number of iterations is determined as 50 for variations of binary SSA equally. The comparison results are shown in Table 16.

In Table 16, the standard deviation, the mean and the best results of variations of binary SSA's superior performance are marked in bold. The variations of binary SSA have displayed an extremely good performance in selected benchmark functions in various comparison criteria according to comparison results. According to the standard deviation results, binary SSA has shown better performance than BPSO, BBA, BSCA and BSalpSA in all the unimodal and multimodal benchmark functions. According to the standard deviation, mean and best results, the binary SSA shows success at roughly 88.89% of multimodal benchmark functions for 8 out of 9 benchmark functions ($f1, f3, f4, f5, f6, f15, f16$ and $f17$). BSalpSA is the second-best performing algorithm after binary SSA. The results

show that binary SSA finds superior solutions with satisfactory standard deviation values. Binary SSA has achieved this success thanks to its similarity measurement and logic gate methods. Thus, the exploration and exploitation capability of binary SSA in the binary search space has been improved. Thus, it is more advantages than other comparison algorithms.

Case Study 2 Social spider optimization is another comparative heuristic algorithm that is very similar to the SSA algorithm, but totally different from it. In SSO, there are two different searching agents (spider) in the search space of SSO. These are male and female spiders. Depending upon the gender, each individual operates a range of different evolutionary operators that imitates various collaboration behaviors that typically exist in colonies (Cuevas et al. 2013; Shukla and Nanda 2018). In SSA, there is no gender discrimination between the spiders. There is a single type searching agent (spider) in search space. In this paper, BSSO and binary SSA are also compared in various unimodal and multimodal benchmark functions in equal comparison parameters. Table 17 shows the parameters for both algorithms. The results of the best, mean, worst and standard deviation are shown in Tables 18

Table 16 Variations of binary SSA and its comparison with comparative algorithms in BPSO, BBA, BSCA and BSalpSA

| Function | BPSO | BBA | BSCA | BSalpSA | Binary SSA-Tanh | Binary SSA-Sigm | Binary SSA-MSigm | Binary SSA-Arctan |
|------------|----------------|------------------|------------|------------------|-----------------|-----------------|------------------|-------------------|
| <i>f1</i> | | | | | | | | |
| Best | 1.99E+09 | 1.76E+08 | 5.58E+09 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Mean | 3.42E+09 | 3.01E+08 | 7.82E+09 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Worst | 4.45E+09 | 5E+08 | 8.69E+09 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| SD | 5.41E+08 | 97.588.6050 | 7.56E+08 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| <i>f3</i> | | | | | | | | |
| Best | 39,863.33 | 2278.4530 | 85,869.500 | 7.5000 | 0.0000 | 6.0001 | 7.5000 | 0.0000 |
| Mean | 51,400.34 | 5407.4590 | 112,325.20 | 7.5000 | 0.0000 | 6.0001 | 7.5000 | 0.0000 |
| Worst | 60,413.45 | 7471.4220 | 138,006.70 | 7.5000 | 0.0000 | 6.0001 | 7.5000 | 0.0000 |
| SD | 5783.194 | 1488.3310 | 15,623.490 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| <i>f4</i> | | | | | | | | |
| Best | 2.55E+09 | 21,957,400 | 1.01E+10 | 3.48E-05 | 1.26E-05 | 4.26E-04 | 4.250E-04 | 1.22E-05 |
| Mean | 3.33E+09 | 70,533,281 | 1.28E+10 | 0.000925 | 1.35E-04 | 5.35E-02 | 4.52E-02 | 1.85E-04 |
| Worst | 4.17E+09 | 1.39E+08 | 1.5E+10 | 0.001824 | 1.57E-03 | 7.56E-03 | 6.85E-03 | 2.77E-03 |
| SD | 7.21E+08 | 49613853 | 2.33E+09 | 0.0007 | 0.0006 | 0.0007 | 0.0007 | 0.0006 |
| <i>f5</i> | | | | | | | | |
| Best | 79.375 | 41.5000 | 102.5000 | 0.0000 | 0.0000 | 68.0050 | 55.0400 | 0.0000 |
| Mean | 90.9125 | 54.4188 | 112.1125 | 0.0000 | 1.0100 | 75.0900 | 60.7580 | 0.0000 |
| Worst | 100.25 | 67.6250 | 118.5000 | 0.0000 | 6.0000 | 85.0600 | 65.7700 | 0.0000 |
| SD | 6.2326 | 7.5482 | 4.2774 | 0.0000 | 2.1200 | 10.0080 | 3.0020 | 0.0000 |
| <i>f6</i> | | | | | | | | |
| Best | 949.8750 | 254.8750 | 1565.125 | 0.0000 | 0.0000 | 62.4250 | 56.7450 | 0.0000 |
| Mean | 1174.231 | 330.4313 | 1.37E+46 | 0.0000 | 0.0000 | 63.2500 | 66.9800 | 0.0000 |
| Worst | 1354.500 | 452.5000 | 1.36E+47 | 0.0000 | 0.0000 | 68.1100 | 87.5500 | 0.0000 |
| SD | 108.9784 | 53.8048 | 3.34E+46 | 0.0000 | 0.0000 | 5.4780 | 11.2540 | 0.0000 |
| <i>f15</i> | | | | | | | | |
| Best | 3.44E+09 | 1.4E+09 | 7.3E+09 | 0.0000 | 0.0000 | 31.0540 | 0.0000 | 0.0000 |
| Mean | 4.29E+09 | 3E+09 | 8.71E+09 | 0.0000 | 25.0050 | 41.0010 | 29.0410 | 0.0000 |
| Worst | 5.23E+09 | 4.4E+09 | 1.02E+10 | 0.0000 | 47.0140 | 49.0140 | 38.0740 | 0.0000 |
| SD | 5.67E+08 | 7.8E+08 | 7.51E+08 | 0.0000 | 75.1000 | 55.1000 | 95.1000 | 0.0000 |
| <i>f16</i> | | | | | | | | |
| Best | 876,800.5 | 304,285.6 | 1,678,091 | 0.0000 | 0.0000 | 529.4650 | 365.7485 | 0.0000 |
| Mean | 1,078,648 | 488,418.4 | 2,249,554 | 0.0000 | 0.0000 | 580.4650 | 368.7480 | 0.2547 |
| Worst | 1,355,679 | 775,537.3 | 2,494,999 | 0.0000 | 0.0000 | 600.0000 | 524.7840 | 0.4326 |
| SD | 130,769.2 | 130,750.7 | 202,932.8 | 0.0000 | 0.0000 | 233.4580 | 324.5740 | 0.0968 |
| <i>f17</i> | | | | | | | | |
| Best | 21.0415 | 20.4844 | 21.1001 | 20.0517 | 20.0015 | 20.5471 | 20.4175 | 20.5472 |
| Mean | 21.2117 | 20.6155 | 21.1911 | 20.0675 | 20.0478 | 20.9870 | 21.1004 | 20.5147 |
| Worst | 21.3183 | 20.7620 | 21.3143 | 20.1489 | 20.5410 | 21.0141 | 21.9840 | 20.5470 |
| SD | 0.0836 | 0.0824 | 0.0537 | 0.0240 | 0.0147 | 0.0502 | 1.1470 | 0.0287 |
| <i>f18</i> | | | | | | | | |
| Best | - 1182.27 | - 1492.81 | - 1146.07 | - 1499.51 | - 500.0000 | - 421.1470 | - 435.8540 | - 498.0178 |
| Mean | - 1046.53 | - 1114.58 | - 1013.75 | - 1066.95 | - 468.1247 | - 395.1240 | - 400.2140 | - 489.6045 |
| Worst | - 948.489 | - 996.391 | - 859.003 | - 814.186 | - 250.7480 | - 120.0124 | - 80.1450 | - 330.5480 |
| SD | 72.5534 | 141.134 | 103.7176 | 203.8991 | 250.4012 | 240.1204 | 410.2100 | 140.2140 |

Table 17 Parameter settings for BSSO and binary SSA

| Key | Method Name |
|-----------------|---|
| BSSO | $f_{pl} = 0.65$ (lower female percent); $f_{pu} = 0.9$ (upper female percent) |
| Binary SSA | $r_a = 1$; $p_c = 0.7$; $p_m = 0.1$; $ST = 0.3$ |
| Common settings | Population size (N) = 30; number of runs = 20; maximum iterations = 50 |

and 19 for unimodal and multimodal benchmarks, respectively. According to the mean of fitness results, binary SSA shows a higher success than BSSO. According to the standard deviation of fitness results, the binary SSA shows high success at roughly 94.44% of benchmark functions for 17 benchmarks out of 18 benchmarks ($f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12, f13, f14, f15, f16$ and $f18$) and the BSSO shows success at roughly 38.89% of benchmark functions for 7 benchmarks out of 18 benchmarks ($f1, f3, f5, f7, f10, f13$ and $f17$). According to the mean of fitness results, the BSSO shows success at roughly 22.22% of benchmark functions for 4 benchmarks out of 18 benchmarks ($f1, f5, f10$ and $f17$). Thanks to the logic gate and similarity measurement techniques, binary SSA has passed BSSO. It has developed the ability to search locally and search globally in binary space.

Case Study 3: Other meta-heuristic algorithms which are BHPSOWM, PSO, GA and variations of binary SSA are compared according to four different criteria. These are the mean, the standard deviation (SD), the best and the worst criteria. In the compared algorithms, the population size (N) is determined as 30 and the maximum number of iterations is determined as 50 for variations of binary SSA equally. The comparison results are shown in Table 20. The standard deviation, the mean and the best results of variations of binary SSA's superior performance are marked in bold. According to the standard deviation and mean results, binary SSA has shown better performance than BHPSOWM, PSO and GA in all the unimodal and multimodal benchmark functions. The SimSSA and LogicSSA techniques have a significant impact on the performance of binary SSA. Based on the obtained results, it is concluded that the proposed binary SSA has promising results for BOPs.

4.5 Discussion about the variations of binary SSA

In this paper, the SSA is modified again according to four different transfer functions for mapping the continuous search space to the binary search space. Four variations of binary SSA are proposed based on four transfer functions. In order to ensure the balance between exploration and exploitation, two different techniques (SimSSA and LogicSSA) are developed at the candidate solution production stage. Performances of the variations of binary SSA are

tested with eighteen benchmark functions which are widely used unimodal and multimodal benchmark functions in the literature. The variations of binary SSA are compared according to four different criteria. There are the mean, the standard deviation (SD), the best and the worst criteria. Performances of the variations of binary SSA are also compared with well-known and recently developed binary methods in the literature.

$f2, f7, f8, f9, f10, f11, f12, f13$ and $f14$ benchmark functions are also used as a test function for variations of binary SSA, and the obtained results are compared with variations of binary SSA. But, related performance comparisons could not be performed and are not mentioned in this paper as there are not many papers that are studied with the benchmark functions in the literature.

According to the comparisons among the variations of binary SSA, we can state that binary SSA-Tanh and the binary SSA-Arctan variations have high performance among other variations of binary SSA and comparative algorithms. In this context, the advantages behind this high performance are due to one reason: Binary SSA combines the methods of SimSSA and LogicSSA which can make the balance between exploration and exploitation capabilities. The transfer function is supported by these methods. Thus, binary SSA's binary optimization problem-solving success is increased.

5 Conclusion

Social spider algorithm (SSA) is a heuristic algorithm that is created by imitating spider behaviors in nature. In this paper, SSA has studied details and a binary SSA has been proposed to solve binary optimization problems (BOPs). In order to obtain binary search space in binary SSA, four different transfer functions are used. Four different variations of binary SSA (binary SSA-Tanh, binary SSA-Sigm, binary SSA-MSigm and binary SSA-Arctan) have been developed. Two new techniques (SimSSA and LogicSSA) are developed during the candidate solutions production phase in binary SSA, and the variations of binary SSA based on SimSSA and LogicSSA are developed. The SimSSA technique uses Jaccard's similarity in a new spider (candidate solution) production phase, and the LogicSSA technique uses consist of XOR logic gate in the new spider (candidate solution) production phase. In the binary SSA,

Table 18 Variations of binary SSA and its comparison with BSSO in all the unimodal benchmark functions

| Function | Binary SSA-Tanh | Binary SSA-Sigm | Binary SSA-MSigm | Binary SSA-Arctan | BSSO |
|------------|-----------------|------------------|------------------|-------------------|---------------|
| <i>f1</i> | | | | | |
| Best | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Mean | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Worst | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| SD | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| <i>f2</i> | | | | | |
| Best | 0.0000 | 2.0000 | 2.0000 | 0.0000 | 0.0000 |
| Mean | 1.0124 | 2.0410 | 2.1045 | 1.0214 | 3.1020 |
| Worst | 1.5600 | 2.5600 | 2.3600 | 1.1500 | 5.2400 |
| SD | 0.0280 | 0.2800 | 0.4500 | 0.0250 | 0.3250 |
| <i>f3</i> | | | | | |
| Best | 0.0000 | 6.0001 | 7.5000 | 0.0000 | 1.0579 |
| Mean | 0.0000 | 6.0001 | 7.5000 | 0.0000 | 1.0579 |
| Worst | 0.0000 | 6.0001 | 7.5000 | 0.0000 | 1.0579 |
| SD | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| <i>f4</i> | | | | | |
| Best | 1.26E-05 | 4.26E-04 | 4.25E-04 | 1.22E-05 | 4.50E-03 |
| Mean | 1.35E-04 | 5.35E-02 | 4.52E-02 | 1.85E-04 | 1.74E-02 |
| Worst | 1.57E-03 | 7.56E-03 | 6.85E-03 | 2.77E-03 | 7.14E-04 |
| SD | 0.0006 | 0.0007 | 0.0007 | 0.0006 | 0.0084 |
| <i>f5</i> | | | | | |
| Best | 0.0000 | 68.0050 | 55.0400 | 0.0000 | 0.0000 |
| Mean | 1.0100 | 75.0900 | 60.7580 | 0.0000 | 0.0000 |
| Worst | 6.0000 | 85.0600 | 65.7700 | 0.0000 | 0.0000 |
| SD | 2.1200 | 10.0080 | 3.0020 | 0.0000 | 0.0000 |
| <i>f6</i> | | | | | |
| Best | 0.0000 | 62.4250 | 56.7450 | 0.0000 | 1.4428 |
| Mean | 0.0000 | 63.2500 | 66.9800 | 0.0000 | 1.8501 |
| Worst | 0.0000 | 68.1100 | 87.5500 | 0.0000 | 2.5410 |
| SD | 0.0000 | 5.4780 | 11.2540 | 0.0000 | 0.0475 |
| <i>f7</i> | | | | | |
| Best | 0.0000 | 4.0001 | 8.0127 | 0.0000 | 9.1710 |
| Mean | 13.0017 | 17.0127 | 15.1027 | 8.4150 | 9.1710 |
| Worst | 18.0020 | 35.0002 | 29.0001 | 14.0000 | 9.1710 |
| SD | 0.0129 | 0.1441 | 0.0210 | 0.0128 | 0.0000 |
| <i>f8</i> | | | | | |
| Best | 0.0000 | - 15.0001 | - 15.0001 | - 15.0001 | 0.8830 |
| Mean | 4.0174 | - 12.0014 | - 10.0042 | - 6.4710 | 9.1240 |
| Worst | 10.0000 | 15.0001 | 15.0001 | 12.0002 | 15.0002 |
| SD | 0.3340 | 0.3210 | 0.3220 | 0.3240 | 15.4100 |
| <i>f9</i> | | | | | |
| Best | 0.0000 | 1.00E-05 | 0.0000 | 0.0000 | 1.2386 |
| Mean | 2.1700 | 7.1700 | 0.0000 | 0.0000 | 8.1900 |
| Worst | 10.0000 | 10.0000 | 0.0000 | 0.0000 | 9.1900 |
| SD | 3.6704 | 4.8448 | 0.0000 | 0.0000 | 3.4750 |
| <i>f10</i> | | | | | |
| Best | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Mean | 0.0000 | 0.5000 | 0.0000 | 0.0000 | 0.0000 |

Table 18 (continued)

| Function | Binary SSA-Tanh | Binary SSA-Sigm | Binary SSA-MSigm | Binary SSA-Arctan | BSSO |
|------------|-----------------|-----------------|------------------|-------------------|---------------|
| Worst | 0.0000 | 1.0000 | 0.0000 | 0.0000 | 0.0000 |
| SD | 0.0000 | 0.5000 | 0.0000 | 0.0000 | 0.0000 |
| <i>f11</i> | | | | | |
| Best | 0.0000 | 20.0000 | 0.0000 | 0.0000 | 0.0562 |
| Mean | 1.3300 | 35.2000 | 1.00E-05 | 2.50E-05 | 27.2000 |
| Worst | 8.0001 | 65.0000 | 4.00E-05 | 9.00E-05 | 40.2000 |
| SD | 2.9800 | 13.0000 | 1.53E-05 | 3.25E-05 | 21.0172 |

The best results are marked with bold

Table 19 Variations of binary SSA and its comparison with BSSO in all the multimodal benchmark functions

| Function | Binary SSA-Tanh | Binary SSA-Sigm | Binary SSA-MSigm | Binary SSA-Arctan | BSSO |
|------------|-------------------|-------------------|-------------------|--------------------|---------------|
| <i>f12</i> | | | | | |
| Best | 4.0000 | 3.0000 | 1.0000 | 17.0001 | 0.1999 |
| Mean | 35.7000 | 64.8000 | 40.3000 | 54.0000 | 66.9000 |
| Worst | 80.0010 | 100.0000 | 91.0001 | 86.0001 | 92.9000 |
| SD | 24.0040 | 30.0745 | 29.7247 | 22.2411 | 30.1240 |
| <i>f13</i> | | | | | |
| Best | - 1.27E+02 | - 1.27E+02 | - 1.27E+02 | - 1.27E+02 | - 17.4184 |
| Mean | - 1.95E+01 | - 5.75E+01 | - 1.24E+02 | - 1.27E+02 | - 17.4184 |
| Worst | 1.27E+02 | - 1.25E+02 | - 1.11E+02 | - 1.27E+02 | - 17.4184 |
| SD | 67.8939 | 85.9569 | 5.9629 | 0.0000 | 0.0000 |
| <i>f14</i> | | | | | |
| Best | 0.0000 | - 15.0001 | - 15.0002 | - 15.0002 | 7.94E-04 |
| Mean | 2.8000 | - 3.33E-01 | - 15.000117 | - 15.000200 | 7.4800 |
| Worst | 6.0000 | 14.0000 | - 15.0001 | - 15.0002 | 16.9000 |
| SD | 2.3152 | 10.8884 | 3.73E-05 | 1.78E-15 | 4.4780 |
| <i>f15</i> | | | | | |
| Best | 0.0000 | 31.0540 | 0.0000 | 0.0000 | 1.8700 |
| Mean | 25.0050 | 41.0010 | 29.0410 | 0.0000 | 2.6700 |
| Worst | 47.0140 | 49.0140 | 38.0740 | 0.0000 | 3.0001 |
| SD | 75.1000 | 55.1000 | 95.1000 | 0.0000 | 15.2140 |
| <i>f16</i> | | | | | |
| Best | 0.0000 | 529.4650 | 365.7485 | 0.0000 | 8.59E-04 |
| Mean | 0.0000 | 580.4650 | 368.7480 | 0.2547 | 3.68E+02 |
| Worst | 0.0000 | 600.0000 | 0524.7840 | 0.4326 | 5.69E+02 |
| SD | 0.0000 | 233.4580 | 324.5740 | 0.0968 | 286.1200 |
| <i>f17</i> | | | | | |
| Best | 20.0015 | 20.5471 | 20.4175 | 20.5472 | 1.2013 |
| Mean | 20.0478 | 20.9870 | 21.1004 | 20.5147 | 1.2013 |
| Worst | 20.5410 | 21.0141 | 21.9840 | 20.5470 | 1.2013 |
| SD | 0.0147 | 0.0502 | 1.1470 | 0.0287 | 0.0000 |
| <i>f18</i> | | | | | |
| Best | - 500.0000 | - 421.1470 | - 435.8540 | - 498.0178 | - 300.1407 |
| Mean | - 468.1247 | - 395.1240 | - 400.2140 | - 489.6045 | - 259.1470 |
| Worst | - 250.7480 | - 120.0124 | - 80.1450 | - 330.5480 | - 17.5120 |
| SD | 250.4012 | 240.1204 | 410.2100 | 140.2140 | 300.1420 |

The best results are marked with bold

Table 20 Variations of binary SSA and its comparison with comparative algorithms in BHPsOWM, PSO and GA

| Function | BHPsOWM | PSO | GA | Binary SSA-Tanh | Binary SSA-Sigm | Binary SSA-MSigm | Binary SSA-Arctan |
|------------|---------------|---------------|---------------|-------------------|-----------------|------------------|-------------------|
| <i>f1</i> | | | | | | | |
| Best | 0.0000 | 312.1800 | 216.3200 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Mean | 42.3960 | 612.2510 | 487.2180 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Worst | – | – | – | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| SD | 52.1120 | 204.0760 | 175.8240 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| <i>f3</i> | | | | | | | |
| Best | 7.5000 | 7.8430 | 7.7590 | 0.0000 | 6.0001 | 7.5000 | 0.0000 |
| Mean | 7.6140 | 8.6110 | 8.3570 | 0.0000 | 6.0001 | 7.5000 | 0.0000 |
| Worst | – | – | – | 0.0000 | 6.0001 | 7.5000 | 0.0000 |
| SD | 0.0670 | 0.3530 | 0.3380 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| <i>f4</i> | | | | | | | |
| Best | 1.740 0 | 1.8000 | 1.7270 | 1.26E–05 | 4.26E–04 | 4.250E–04 | 1.22E–05 |
| Mean | 2.1150 | 2.0300 | 1.9810 | 1.35E–04 | 5.35E–02 | 4.52E–02 | 1.85E–04 |
| Worst | – | – | – | 1.57E–03 | 7.56E–03 | 6.85E–03 | 2.77E–03 |
| SD | 0.1110 | 0.0900 | 0.1120 | 0.0006 | 0.0007 | 0.0007 | 0.0006 |
| <i>f5</i> | | | | | | | |
| Best | 20.8000 | 16.0000 | 11.3000 | 0.0000 | 68.0050 | 55.0400 | 0.0000 |
| Mean | 25.2760 | 26.5240 | 14.1710 | 1.0100 | 75.0900 | 60.7580 | 0.0000 |
| Worst | – | – | – | 6.0000 | 85.0600 | 65.7700 | 0.0000 |
| SD | 1.3890 | 2.1860 | 1.2810 | 2.1200 | 10.0080 | 3.0020 | 0.0000 |
| <i>f6</i> | | | | | | | |
| Best | 0.0000 | 5.4200 | 4.3200 | 0.0000 | 62.4250 | 56.7450 | 0.0000 |
| Mean | 2.1460 | 10.8530 | 8.5200 | 0.0000 | 63.2500 | 66.9800 | 0.0000 |
| Worst | – | – | – | 0.0000 | 68.1100 | 87.5500 | 0.0000 |
| SD | 0.8130 | 2.5970 | 2.2780 | 0.0000 | 5.4780 | 11.2540 | 0.0000 |
| <i>f15</i> | | | | | | | |
| Best | 25.2790 | 115.5050 | 55.8390 | 0.0000 | 31.0540 | 0.0000 | 0.0000 |
| Mean | 76.182 0 | 275.2180 | 226.6330 | 25.0050 | 41.0010 | 29.0410 | 0.0000 |
| Worst | – | – | – | 47.0140 | 49.0140 | 38.0740 | 0.0000 |
| SD | 26.7490 | 78.3830 | 84.4340 | 75.1000 | 55.100 | 95.1000 | 0.0000 |
| <i>f16</i> | | | | | | | |
| Best | 10 (– 9) | 10 (– 9) | 10 (– 9) | 0.0000 | 529.4650 | 365.7485 | 0.0000 |
| Mean | 10 (– 9) | 10 (– 9) | 10 (– 9) | 0.0000 | 580.4650 | 368.7480 | 0.2547 |
| Worst | – | – | – | 0.0000 | 600.0000 | 524.7840 | 0.4326 |
| SD | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 233.4580 | 324.5740 | 0.0968 |
| <i>f17</i> | | | | | | | |
| Best | 1.7180 | 1.7180 | 1.7180 | 20.0015 | 20.5471 | 20.4175 | 20.5472 |
| Mean | 1.7180 | 1.7180 | 1.7180 | 20.0478 | 20.9870 | 21.1004 | 20.5147 |
| Worst | – | – | – | 20.5410 | 21.0141 | 21.9840 | 20.5470 |
| SD | 0.0000 | 0.0000 | 0.0000 | 0.0147 | 0.0502 | 1.1470 | 0.0287 |
| <i>f18</i> | | | | | | | |
| Best | 946.4870 | 1153.4540 | 1210.0810 | – 500.0000 | – 421.1470 | – 435.8540 | – 498.0178 |
| Mean | 1984.8750 | 1960.3150 | 2072.1080 | – 468.1247 | – 395.1240 | – 400.2140 | – 489.6045 |
| Worst | – | – | – | – 250.7480 | – 120.0124 | – 80.1450 | – 330.5480 |
| SD | 267.4130 | 401.9830 | 472.0120 | 250.4012 | 240.1204 | 410.2100 | 140.2140 |

one of the SimSSA or LogicSSA techniques is selected for the production of new candidate solutions at any time. Binary SSA's local search capability has been improved with logic gates that are frequently used in the BOPs in recent years. With the similarity measurement technique, the power of binary SSA to discover new points in binary search space is increased. With the similarity measurement technique, this algorithm guarantees to find global optimum instead of local optimums. The variations of binary SSA are operated in eighteen different unimodal and multimodal benchmark functions in order to evaluate their performances. The performance of variations of binary SSA is compared in various criteria. Wilcoxon signed-rank test is operated on obtained results. Particle swarm optimization (PSO), genetic algorithm (GA), BSalpSA, BHPSOWM, BPSO, BBA, BSCA and BSSO which are well-known heuristics algorithms in the literature are compared with the variations of binary SSA.

According to the comparisons among the variations of binary SSA, we can state that binary SSA-Arctan and binary SSA-Tanh variations have high performance among other variations of binary SSA. The comparison results affirmed the superiority of the binary SSA-Arctan and binary SSA-Tanh variations in providing a good quality of solutions for most tests.

In future research, we will design new transfer functions. Besides, we will measure the performance of our proposed method in different applications including knapsack problems and feature selection in classification.

5.1 Replication of results

Most of the codes required to replicate the results in this paper are available under open-source licenses and are maintained in version control repositories. The spider social algorithm and the spider social optimization are available from GitHub (<https://github.com/James-Yu/SocialSpiderAlgorithm>). Benchmark test functions are available from <http://www.sfu.ca/ssurjano>.

Compliance with ethical standards

Conflict of interest There is no conflict of interest between the authors to publish this manuscript.

References

- Acılar AM (2013) Yapay Bağışıklık Algoritmaları Kullanılarak Bulanık Sistem Tasarımı, Konya, Turkey. Ph.D. thesis (in Turkish)
- Aslan M, Gunduz M, Kiran MS (2019) JayaX: Jaya algorithm with xor operator for binary optimization. *Appl Soft Comput J* 82:105576
- Babaoglu I, Findik O, Ulker E (2010) A comparison of feature selection models utilizing binary particle swarm optimization and genetic algorithm in determining coronary artery disease using support vector machine. *Expert Syst Appl* 37:3177–3183
- Beskirli M, Koc I, Hakli H, Kodaz H (2018) A new optimization algorithm for solving wind turbine placement problem: binary artificial algae algorithm. *Renew Energy* 121:301–308
- Choi SS, Cha SH, Tappert CC (2010) A survey of binary similarity and distance measures. *J Syst Cybern Inform* 8(1):43–48
- Çınar AC, Kiran MS (2018) Similarity and logic gate-based tree-seed algorithms for binary optimization. *Comput Ind Eng* 115:631–646
- Cuevas E, Cienfuegos M, Zaldívar D, Pérez-Cisneros M (2013) A swarm optimization algorithm inspired in the behavior of the social-spider. *Expert Syst Appl* 40:6374–6384
- Derrac J, García S, Molina D, Herrera F (2011) A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol Comput* 1:3–18
- Dorigo M (1990) Optimization learning and natural algorithms. Politecnico di Milano, Italie, Ph.D. thesis
- El-Bages MS, Elsayed WT (2017) Social spider algorithm for solving the transmission expansion planning problem. *Electr Power Syst Res* 143:235–243
- Elsayed WT, Hegazy YG, Bendary FM, El-Bages MS (2016) Modified social spider algorithm for solving the economic dispatch problem. *Eng Sci Technol Int J* 19:1672–1681
- Emary E, Zawbaa HM, Hassani AE (2016) Binary grey wolf optimization approaches for feature selection. *Neurocomputing* 172:371–381
- Fan K, Weijia Y, Li Y (2013) An effective modified binary particle swarm optimization (mBPSO) algorithm for multi-objective resource allocation problem (MORAP). *Appl Math Comput* 221:257–267
- Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
- Jaccard P (1901) Etude comparative de la distribution florale dans une portion des Alpes et du Jura: Impr. Corbaz
- Jamil M, Yang XS (2013) A literature survey of benchmark functions for global optimization problems. *Int J Math Model Numer Optim* 4:150–194
- Jiang F, Xia H, Tran QA, Ha QM, Tran NQ, Hu J (2017) A new binary hybrid particle swarm optimization with wavelet mutation. *Knowl Based Syst* 130:90–101
- Karaboga D (2005) An idea based on honey bee swarm for numerical optimization. Erciyes University, Engineering Faculty, Computer Engineering Department, Kayseri, Turkey, pp 1–10. Ph.D. thesis (in Turkish)
- Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of IEEE international conference on neural networks, Perth, WA, pp 1942–1948
- Kennedy J, Eberhart R (1997) A discrete binary version of the particle swarm algorithm. In: Proceedings of the IEEE international conference on computational cybernetics and simulation. <https://doi.org/10.1109/icsmc.1997.637339>
- Kiran MS, Gunduz M (2013) XOR-based artificial bee colony algorithm for binary optimization. *Turk J Electr Eng Comput Sci* 21:2307–2328
- Korkmaz S, Babalik A, Servet KM (2017) An artificial algae algorithm for solving binary optimization problems. *J Mach Learn Cybern, Int.* <https://doi.org/10.1007/s13042-017-0772-7>
- Kurt M, Semetay C (2001) Genetik Algoritma ve Uygulama Alanları. *Turk J Mühendis ve Makina* 42(501):19–24 (in Turkish)

- Ling W, Fu X, Menhas M, Fei M (2010a) A modified binary differential evolution algorithm. In: Li K, Fei M, Jia L, Irwin GW (eds) *Life system modeling and intelligent computing*, vol 6329. Springer, Berlin, pp 49–57
- Ling W, Xu Y, Mao Y, Fei M (2010b) A discrete harmony search algorithm. In: Li K, Fei M, Jia L, Irwin GW (eds) *Life system modeling and intelligent computing*, vol 98. Springer, Berlin, pp 37–43
- Mallipeddi R, Mallipeddi S, Suganthan PN, Tasgetiren MF (2011) Differential evolution algorithm with ensemble of parameters and mutation strategies. *Appl Soft Comput* 11:1679–1696
- Mirjalili S, Lewis A (2013) S-shaped versus V-shaped transfer functions for binary particle swarm optimization. *Swarm Evol Comput* 9:1–14
- Mirjalili S, Mohd Hashim SZ (2012) BMOA: binary magnetic optimization algorithm. *Int J Mach Learn Comput* 2(3):204–208
- Mousa A, Bentahar J (2016) An efficient QoS-aware web services selection using social spider algorithm. In: *The 13th international conference on mobile systems and pervasive Computing (MobiSPC 2016)*, *procedia computer science*, vol 94, pp 176–182
- Omkar S, Senthilnath J, Khandelwal R, Naik GN, Gopalakrishnan S (2011) Artificial bee colony (ABC) for multi-objective design optimization of composite structures. *Appl Soft Comput* 11:489–499
- Pal A, Maiti J (2010) Development of a hybrid methodology for dimensionality reduction in Mahalanobis–Taguchi system using Mahalanobis distance and binary particle swarm optimization. *Expert Syst Appl* 37:1286–1293
- Parpinelli RS, Lopes HS (2011) New inspirations in swarm intelligence: a survey. *Int J Bio-Inspired Comput* 3(1):1–16
- Pereira LAM, Rodrigues D, Ribeiro PB, Papa JP, Weber SAT (2014) Social-spider optimization-based artificial neural networks training and its applications for Parkinson’s disease identification. In: *2014 IEEE 27th international symposium on computer-based medical systems*, pp 14–17
- Prescilla K, Immanuel SA (2013) Modified binary particle swarm optimization algorithm application to real-time task assignment in a heterogeneous multiprocessor. *Microprocess Microsyst* 37:583–589
- Qiao LY, Peng XY, Peng Y (2006) BPSO-SVM wrapper for feature subset selection. *Dianzi Xuebao (Acta Electron Sin)* 34:496–498
- Rashedi E, Nezamabadi-pour H, Saryazdi S (2009) BGSA: binary gravitational search algorithm. *Nat Comput* 9(3):727–745
- Rizk-Allah RM (2014) A novel multi-ant colony optimization for multi-objective resource allocation problems. *Int J Math Arch* 5:183–192
- Rizk-Allah RM (2018) Hybridizing sine cosine algorithm with a multi-orthogonal search strategy for engineering design problems. *J Comput Des Eng* 5:249–273
- Rizk-Allah RM, Hassanién AE (2018) New binary bat algorithm for solving 0–1 knapsack problem. *Complex Intell Syst* 4:31–53
- Rizk-Allah RM, Hassanién AE, Elhoseny M, Gunasekaran M (2018) A new binary salp swarm algorithm: development and application for optimization tasks. *Neural Comput Appl*. <https://doi.org/10.1007/s00521-018-3613>
- Shukla UP, Nanda SJ (2018) A binary social spider optimization algorithm for unsupervised band selection in compressed hyperspectral images. *Expert Syst Appl* 97:336–356
- Sun S, Qi H, Sun Jianping, Ren Y, Ruan L (2017) Estimation of thermophysical properties of phase change material by the hybrid SSO algorithms. *Int J Therm Sci* 120:121–135
- Surjanovic S, Bingham D (2019) Virtual library of simulation experiments: test functions and datasets. <http://www.sfu.ca/ssurjano>
- Talbi EG (2009) *Metaheuristics: from design to implementation*. Wiley, Hoboken
- Yu JJQ, Li VOK (2015) A social spider algorithm for global optimization. *Appl Soft Comput* 30:614–627
- Yu JJQ, Li VOK (2016) A social spider algorithm for solving the non-convex economic load dispatch problem. *Neurocomputing* 171(C):955–965

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.