Applied Soft Computing 30 (2015) 614–627

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A social spider algorithm for global optimization

James J.Q. Yu*, Victor O.K. Li

Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam, Hong Kong

ARTICLE INFO

Article history:
Received 19 August 2014
Received in revised form 23 January 2015
Accepted 9 February 2015
Available online 17 February 2015

Keywords:
Social spider algorithm
Global optimization
Swarm intelligence
Evolutionary computation
Meta-heuristic

ABSTRACT

The growing complexity of real-world problems has motivated computer scientists to search for efficient problem-solving methods. Metaheuristics based on evolutionary computation and swarm intelligence are outstanding examples of nature-inspired solution techniques. Inspired by the social spiders, we propose a novel social spider algorithm to solve global optimization problems. This algorithm is mainly based on the foraging strategy of social spiders, utilizing the vibrations on the spider web to determine the positions of preys. Different from the previously proposed swarm intelligence algorithms, we introduce a new social animal foraging strategy model to solve optimization problems. In addition, we perform preliminary parameter sensitivity analysis for our proposed algorithm, developing guidelines for choosing the parameter values. The social spider algorithm is evaluated by a series of widely used benchmark functions, and our proposed algorithm has superior performance compared with other state-of-the-art metaheuristics.

1. Introduction

With the fast growing size and complexity of modern optimization problems, evolutionary computing is becoming increasingly attractive as an efficient tool for optimization. Depending on the nature of phenomenon simulated, evolutionary computing algorithms can be classified into two important groups: evolutionary algorithms (EAs) and swarm intelligence based algorithms. EAs, which mainly draw inspiration from nature, have been shown to be very successful for optimization among all the methods devised by the evolutionary computation community. Currently several types of EAs have been widely employed to solve real world combinatorial or global optimization problems, including genetic algorithm (GA), genetic programming (GP), evolutionary strategy (ES) and differential evolution (DE). These algorithms demonstrate satisfactory performance compared with conventional optimization techniques, especially when applied to solve non-convex optimization problems [1,2].

In the past two decades, swarm intelligence, a new kind of evolutionary computing technique, has attracted much research interest [3]. The term swarm is employed in a general manner to refer to any collection of interactive agents. Swarm intelligence is mainly concerned with the methodology to model the behavior of social animals and insects for problem solving. Researchers devised optimization algorithms by mimicking the behavior of ants, bees, bacteria, fireflies and other organisms. The impetus of creating such algorithms was provided by the growing needs to solve optimization problems that were very difficult or even considered intractable.

Among the commonly seen animals, spiders have been a major research subject in bionic engineering for many years. However, most research related to spiders focused on the imitation of its walking pattern to design robots, e.g. [4]. A possible reason for this is that a majority of the spiders observed are solitary [5], which means that they spend most of their lives without interacting with others of their species. However, among the 35 000 spider species observed and described by scientists, some species are social. These spiders, e.g. *Mallos gregalis* and *Oecobius civitas*, live in groups and interact with others in the same group. Based on these social spiders, this paper formulates a new global optimization method to solve optimization problems.

Spiders are air-breathing arthropods. They have eight legs and chelicerae with fangs. Spiders have been found worldwide and are one of the most diverged species among all groups of organisms. They use a wide range of strategies for foraging, and most of them detect prey by sensing vibrations. Spiders have long been known to be very sensitive to vibratory stimulation, as vibrations on their webs notify them of the capture of prey. If the vibrations are in a defined range of frequency, spiders attack the vibration source. The social spiders can also distinguish vibrations generated by the prey with ones generated by other spiders [6]. The social spiders passively receive the vibrations generated by other spiders

* Corresponding author. Tel.: +852 64301673.
E-mail addresses: jqyu@eee.hku.hk (J.J.Q. Yu), vli@eee.hku.hk (V.O.K. Li).

http://dx.doi.org/10.1016/j.asoc.2015.02.014
1568-4946/

on the same web to have a clear view of the web. This is one of the unique characteristics which distinguishes the social spiders from other organisms as the latter usually exchange information actively, which reduces the information loss to some degree but increases the energy used per communication action [7].

The group living phenomenon has been studied intensively in animal behavior ecology. One of the reasons that animals gather and live together is to increase the possibility of successful foraging and reduce the energy cost in this process [8]. In order to facilitate the analysis of social foraging behavior, researchers proposed two foraging models: information sharing (IS) model [9] and producer–scrounger (PS) model [10]. The individuals under the IS model perform searching and seek for opportunity to join other individuals simultaneously. In the PS model, the individuals are divided into leaders and followers. Since there is no leader in social spiders [11], the IS model is more suitable to formulate the foraging behavior of social spiders, and we use this model to control the searching pattern of our proposed algorithm.

In this paper, inspired by the social behavior of the social spiders, especially their foraging behavior, we propose a new metaheuristic for global optimization: the social spider algorithm (SSA). The foraging behavior of the social spider can be described as the cooperative movement of the spiders towards the food source position. The spiders receive and analyze the vibrations propagated on the web to determine the potential direction of a food source [12]. We utilize this natural behavior to perform optimization over the search space in SSA.

The contribution of this paper is threefold:

- We propose a new nature-inspired swarm intelligence algorithm based on social spiders. This population-based general-purpose metaheuristic demonstrates outstanding performance in the global optimization benchmark tests.
- We introduce a new social animal foraging model into metaheuristic design. This is the very first attempt of employing the IS model to solve optimization problems. We also incorporate the information loss schemes in the algorithm, which is a unique design of our proposed algorithm.
- We perform a series of experiments to investigate the impact of different parameters and searching schemes on the performance of the algorithm. The result of these experiments may serve as important inputs for further research.

The rest of this paper is organized as follows. We will first present some related work on swarm intelligence and bio-inspired metaheuristics in Section 2. Then we will formulate and elaborate on SSA by idealizing and imitating the foraging behavior of social spiders in Section 3. Section 4 introduces the benchmark functions we use for testing the performance of SSA, with the experimental settings. Section 5 presents the simulation results of SSA on the benchmark functions and the comparison with other popular metaheuristics. Finally we will conclude this paper in Section 6 and propose some future work.

2. Background

Swarm intelligence algorithms mimic the methods in nature to drive a search for the optimal solution. At the very beginning there are two major methods for this kind of algorithms: ant colony optimization (ACO) [13] and particle swarm optimization (PSO) [14].

ACO is inspired by the foraging behavior of ants, whose goal is to find a shortest path from their colony to food sources. In this metaheuristic, feasible solutions of the optimization problem to be solved are represented by the paths between the colony and food sources. The ants communicate with and influence others using pheromone, a volatile chemical substance. When an ant finds a food source, it deposits certain amount of pheromone along the path and the amount is positively correlated with the quality of the food source. The pheromone laid down biases the path selection of other ants, providing positive feedback. Using the scheme of positive feedback, the algorithm leads the ants to find the shortest path to a best food source [13].

PSO is motivated by the movement of organisms as a group, as in a flock of birds or a school of fishes. The group is represented by a swarm of particles and PSO uses their positions in the search space to represent the feasible solutions of the optimization problem. PSO manipulates the movement of these particles to perform optimization, utilizing the information of individual experience and socio-cognitive tendency. These two kinds of information correspond to cognitive learning and social learning, respectively, and lead the population to find a best way to perform optimization [14].

The above two metaheuristics have been applied to solve a vast range of different problems, e.g. [15,16]. Motivated by such success, swarm intelligence algorithm design has attracted many researchers and several new algorithms were devised. The most widely studied organism in swarm intelligence is the bee [3]. Abbass proposed a Marriage in honey Bees Optimization (MBO) in [17] and this algorithm was applied to solve propositional satisfiability problems (3-SAT problems). In MBO, the mating flight of the queen bee is represented as the transitions in a state space (search space), with the queen probabilistically mating with the drone encountered at each state. The probability of mating is determined by the speed and energy of the queen, and the fitness of the drone. Karaboga and Basturk proposed an Artificial Bee Colony optimization (ABC) in [18]. ABC classifies the bees in a hive into three types: "scout bees" that randomly fly without guidance, "employed bees" that search the neighborhood of their positions, and "onlooker bees" that use the population fitness to select a guiding solution for exploitation. The algorithm balances exploration and exploitation by means of using employed and onlooker bees for local search, and the scout bees for global search. It also demonstrates satisfactory performance in applications [19,20].

Besides the bees, other organisms have also been widely studied [3]. Krishnanand and Ghose proposed a Glow-worm Swarm Optimization (GSO) [21] based on the behavior of the firefly. In GSO, each firefly randomly selects a neighbor according to its luminescence and moves toward it. In general the fireflies are more likely to get interested in others that glow brighter. As the movement is only conducted locally using selective neighbor information, the firefly swarm is able to divide into disjoint subgroups to explore multiple optima. Another firefly-based technique is proposed by Yang et al. [22]. He reformulated the co-movement pattern of fireflies and employed it in optimization. Passino devised a Bacterial Foraging Optimization (BFO) [23] based on the bacterial chemotaxis. In BFO, possible solutions to the optimization problem are represented by a colony of bacteria. It consists of three schemes, i.e., chemotaxis, reproduction, and elimination-dispersal. The exploitation task is performed using the first two schemes and the last one contributes to exploration [24]. Researchers have also devised swarm intelligence algorithms based on other organisms and they can also generate satisfactory optimization performance [3]. To the best of our knowledge, only one spider-inspired metaheuristic aiming at solving optimization problem has been proposed, i.e. the social spider optimization [25] devised by Cuevas et al., which divides the spiders into different genders and mimics the mating behavior for optimization. However, our proposed algorithm is totally different from this algorithm in their biological backgrounds, motivations, implementations, and search behaviors. We will further reveal the differences in Section 3.4.

3. Social spider algorithm

In SSA, we formulate the search space of the optimization problem as a hyper-dimensional spider web. Each position on the web represents a feasible solution to the optimization problem and all feasible solutions to the problem have corresponding positions on this web. The web also serves as the transmission media of the vibrations generated by the spiders. Each spider on the web holds a position and the quality (or fitness) of the solution is based on the objective function, and represented by the potential of finding a food source at the position. The spiders can move freely on the web. However, they cannot leave the web as the positions off the web represent infeasible solutions to the optimization problem. When a spider moves to a new position, it generates a vibration which is propagated over the web. Each vibration holds the information of one spider and other spiders can get the information upon receiving the vibration.

3.1. Spider

The spiders are the agents of SSA to perform optimization. At the beginning of the algorithm, a pre-defined number of spiders are put on the web. Each spider s holds a memory, storing the following individual information:

- The position of s on the web.
- The fitness of the current position of s.
- The target vibration of s in the previous iteration.
- The number of iterations since s has last changed its target vibration.
- The movement that s performed in the previous iteration.
- The dimension mask[1] that s employed to guide movement in the previous iteration.

The first two types of information describe the individual situation of s, while all others are involved in directing s to new positions. The detailed scheme of movement will be elaborated in Section 3.3.

Based on observations, spiders are found to have very accurate senses of vibration. Furthermore, they can separate different vibrations propagated on the same web and sense their respective intensities [11]. In SSA, a spider will generate a vibration when it reaches a new position different from the previous one. The intensity of the vibration is correlated with the fitness of the position. The vibration will propagate over the web and other spiders can sense it. In such a way, the spiders on the same web share their personal information with others to form a collective social knowledge.

3.2. Vibration

Vibration is a very important concept in SSA. It is one of the main characteristics that distinguish SSA from other metaheuristics. In SSA, we use two properties to define a vibration, namely, the source position and the source intensity of the vibration. The source position is defined by the search space of the optimization problem, and we define the intensity of a vibration in the range $[0, +\infty)$. Whenever a spider moves to a new position, it generates a vibration at its current position. We define the position of spider a at time t as $\boldsymbol{P}_a(t)$, or simply as $\boldsymbol{P}_a$ if the time argument is t. We further use $I(\boldsymbol{P}_a, \boldsymbol{P}_b, t)$ to represent the vibration intensity sensed by a spider at position $\boldsymbol{P}_b$ at time t and the source of the vibration is at position $\boldsymbol{P}_a$. With these notations we can thus use $I(\boldsymbol{P}_s, \boldsymbol{P}_s, t)$ to represent the intensity of the vibration generated by spider s at the source position. This vibration intensity at the source position is correlated with the fitness of its position $f(\boldsymbol{P}_s)$, and we define the intensity value as follows:

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

where C is a confidently small constant such that all possible fitness values are larger than C. Please note that we consider minimization problems in this paper. The design of (1) takes the following issues into consideration:

- All possible vibration intensities of the optimization problem are positive.
- The positions with better fitness values, i.e. smaller values for minimization problems, have larger vibration intensities than those with worse fitness values.
- When a solution approaches the global optimum, the vibration intensity would not increase excessively, and cause malfunctioning of the vibration attenuation scheme.

As a form of energy, vibration attenuates over distance. This physical phenomenon is accounted for in the design of SSA. We define the distance between spider a and b as $D(\boldsymbol{P}_a, \boldsymbol{P}_b)$ and we use 1-norm (Manhattan distance) to calculate the distance, i.e.,

$$D(\boldsymbol{P}_a, \boldsymbol{P}_b) = ||\boldsymbol{P}_a - \boldsymbol{P}_b||_1. \tag{2}$$

The standard deviation of all spider positions along each dimension is represented by $\boldsymbol{\sigma}$. With these definitions, we further define the vibration attenuation over distance as follows:

$$I(\boldsymbol{P}_a, \boldsymbol{P}_b, t) = I(\boldsymbol{P}_a, \boldsymbol{P}_a, t) \times \exp\left(-\frac{D(\boldsymbol{P}_a, \boldsymbol{P}_b)}{\overline{\boldsymbol{\sigma}} \times r_a}\right). \tag{3}$$

In the above formula we introduce a user-controlled parameter $r_a \in (0, \infty)$. This parameter controls the attenuation rate of the vibration intensity over distance. The larger r_a is, the weaker the attenuation imposed on the vibration.

3.3. Search pattern

Here we demonstrate the above ideas in terms of an algorithm. There are three phases in SSA: initialization, iteration, and final. These three phases are executed sequentially. In each run of SSA, we start with the initialization phase, then perform searching in an iterative manner, and finally terminate the algorithm and output the solutions found.

In the initialization phase, the algorithm defines the objective function and its solution space. The value for the parameter used in SSA is also assigned. After setting the values, the algorithm proceeds to create an initial population of spiders for optimization. As the total number of spiders remains unchanged during the simulation of SSA, a fixed size memory is allocated to store their information. The positions of spiders are randomly generated in the search space, with their fitness values calculated and stored. The initial target vibration of each spider in the population is set at its current position, and the vibration intensity is zero. All other attributes stored by each spider are also initialized with zeros. This finishes the initialization phase and the algorithm starts the iteration phase, which performs the search with the artificial spiders created.

In the iteration phase, a number of iterations are performed by the algorithm. In each iteration, all spiders on the web move to a new position and evaluate their fitness values. Each iteration can be further divided into the following sub-steps: fitness evaluation, vibration generation, mask changing, random walk, and constraint handling.

The algorithm first calculates the fitness values of all the artificial spiders on different positions on the web, and update the global

[1] The dimension mask is a 0-1 binary vector of length D, where D is the dimension of the optimization problem.

optimum value if possible. The fitness values are evaluated once for each spider during each iteration. Then these spiders generate vibrations at their positions using (1). After all the vibrations are generated, the algorithm simulates the propagation process of these vibrations using (3). In this process, each spider s will receive $|pop|$ different vibrations generated by other spiders where pop is the spider population. The received information of these vibrations include the source position of the vibration and its attenuated intensity. We use V to represent these $|pop|$ vibrations. Upon the receipt of V, s will select the strongest vibration v_s^{best} from V and compare its intensity with the intensity of the target vibration v_s^{tar} stored in its memory. s will store v_s^{best} as v_s^{tar} if the intensity of v_s^{best} is larger, and c_s, or the number of iterations since s has last changed its target vibration, is reset to zero; otherwise, the original v_{tar} is retained and c_s is incremented by one. We use $\boldsymbol{P}_s^i$ and $\boldsymbol{P}_s^{tar}$ to represent the source positions of V and v_{tar}, respectively, and $i = \{1, 2, \cdots, |pop|\}$.

The algorithm then manipulates s to perform a random walk towards v_s^{tar}. Here we utilize a dimension mask to guide the movement. Each spider holds a dimension mask $\boldsymbol{m}$, which is a 0-1 binary vector of length D and D is the dimension of the optimization problem. Initially all values in the mask are zero. In each iteration, spider s has a probability of $1 - p_c{}^{c_s}$ to change its mask where $p_c \in (0, 1)$ is a user-defined attribute that describes the probability of changing mask. If the mask is decided to be changed, each bit of the vector has a probability of p_m to be assigned with a one, and $1 - p_m$ to be a zero. p_m is also a user-controlled parameter defined in (0, 1). Each bit of a mask is changed independently and does not have any correlation with the previous mask. In case all bits are zeros, one random value of the mask is changed to one. Similarly, one random bit is assigned to zero if all values are ones.

After the dimension mask is determined, a new following position $\boldsymbol{P}_s^{fo}$ is generated based on the mask for s. The value of ith dimension of the following position $\boldsymbol{P}_{s,i}^{fo}$ is generated as follows.

$$P_{s,i}^{fo} = \begin{cases} P_{s,i}^{tar} & m_{s,i} = 0 \\ P_{s,i}^{r} & m_{s,i} = 1 \end{cases}, \tag{4}$$

where r is a random integer value generated in $[1, |pop|]$, and $m_{s,i}$ stands for the ith dimension of the dimension mask $\boldsymbol{m}$ of spider s. Here the random number r for two different dimensions with $m_{s,i} = 1$ is generated independently.

With the generated $\boldsymbol{P}_s^{fo}$, s performs a random walk to the position. This random walk is conducted using the following equation.

$$\boldsymbol{P}_s(t+1) = \boldsymbol{P}_s + (\boldsymbol{P}_s - \boldsymbol{P}_s(t-1)) \times r + (\boldsymbol{P}_s^{fo} - \boldsymbol{P}_s) \odot \boldsymbol{R}, \tag{5}$$

where $\odot$ denotes element-wise multiplication and $\boldsymbol{R}$ is a vector of random float-point numbers generated from zero to one uniformly. Before following $\boldsymbol{P}_s^{fo}$, s 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 s approaches $\boldsymbol{P}_s^{fo}$ along each dimension with random factors generated in (0, 1). This random factor for different dimensions is generated independently. After this random walk, s stores its movement in the current iteration for the next iteration. This ends the random walk sub-step.

The final sub-step of the iteration phase is the constraint handling. The spiders may move out of the web during the random walk step, which causes the constraints of the optimization problem to be violated. There are many methods to handle the boundary constraints in the previous literature, and the random approach, absorbing approach, and the reflecting approach are three most widely adopted methods [26]. In this paper we adopt the reflecting approach for constraint handling and produce a boundary-constraint-free position $\boldsymbol{P}_s(t+1)$ by

$$P_{s,i}(t+1) = \begin{cases} (\overline{x_i} - P_{s,i}) \times r & if P_{s,i}(t+1) > \overline{x_i} \\ (P_{s,i} - \underline{x_i}) \times r & if P_{s,i}(t+1) < \underline{x_i} \end{cases}, \tag{6}$$

where $\overline{x_i}$ is the upper bound of the search space in the ith dimension, and $\underline{x_i}$ is the lower bound of the corresponding dimension. r is a random floating point number generated in (0, 1)

The iteration phase loops until the stopping criteria is matched. The stopping criteria can be defined as the maximum iteration number reached, the maximum CPU time used, the error rate reached, the maximum number of iterations with no improvement on the best fitness value, or any other appropriate criteria. After the iteration phase, the algorithm outputs the best solution with the best fitness found. The above three phases constitute the complete algorithm of SSA and its pseudo-code can be found in Algorithm 1.

Algorithm 1. Social spider algorithm

```
Assign values to the parameters of SSA.
Create the population of spiders pop and assign memory for them.
Initialize v_s^tar for each spider.
while stopping criteria not met do
  for each spider s in pop do
    Evaluate the fitness value of s.
    Generate a vibration at the position of s.
  end for
  for each spider s in pop do
    Calculate the intensity of the vibrations V
      generated by all spiders.
    Select the strongest vibration v_s^best from V.
    if The intensity of v_s^best is larger than v_s^tar then
      Store v_s^best as v_s^tar.
    end if
    Update c_s.
    Generate a random number r from [0,1).
    if r > p_c^{c_s} then
      Update the dimension mask m_s.
    end if
    Generate P_s^fo.
    Perform a random walk.
    Address any violated constraints.
  end for
end while
Output the best solution found.
```

3.4. Differences between SSA and other evolutionary computation algorithms

A number of swarm intelligence algorithms have been proposed in the past few decades. Among them, PSO and ACO are the two most widely employed and studied algorithms. SSA may also be classified as a swarm intelligence algorithm, but it has many differences from PSO and ACO, elaborated below.

PSO, like SSA, was originally proposed for solving continuous optimization problems. It was also inspired by animal behavior. However, the first crucial difference between SSA and PSO is in individual following patterns. In PSO, all particles follow a common global best position and their own personal best position. However in SSA, all spiders follow positions constructed by others' current positions and their own historical positions. These following positions are not guaranteed to be visited by the population before, and different spiders can have different following positions. Since the global best position and spiders' current positions differ greatly during most time of the optimization process, these two following patterns lead to different searching behaviors. This may weaken the convergence ability of SSA but can potentially strengthen the

capability of solving multi-modal optimization problems with a great number of local optimums.

Besides the difference in the following pattern, the difference in their biology backgrounds is also very significant. PSO was designed based on the model of coordinated group animal motions of flocks of birds or schools of fishes. This model serves as the design metaphor of PSO. SSA is inspired by the social spider foraging strategy, which belongs to the scope of general social animal searching behavior. We use a general IS model as the design framework. This difference is also a major distinguishing feature of SSA from other proposed algorithms.

A third difference between SSA and the original formulation of PSO is in the information propagation method. In PSO, the information propagation method is neglected, and each particle is assumed to be aware of all the information of the system without loss. Although the information validity ranges are considered in some recent variants of PSO, the information loss characteristic is still a unique feature that distinguishes SSA from PSO variants. In SSA we model the information propagation process through the vibrations on the spider web. This process forms a general knowledge system with information loss. Although there is still no research on how the information loss will impact the social foraging strategy employed in optimization, it is possible that this information loss system partially contributes to the performance improvement of SSA over PSO.

Another difference is that in PSO, the common knowledge of the group is all about the best particle in the system. All remaining particles in the system do not constitute the shared information of the group, which may lead to neglecting some valuable information of the population. In SSA, each spider generates a new piece of information and propagates the information to the whole population. There are also some PSO variants that shares personal best position information with the population, but the main focus of PSO is on the best positions of the individuals and the population. In SSA, the information generated and propagated with the vibrations are the current positions instead of the best-in-history positions, which may differ greatly with the search.

Although both SSA and ACO draw their inspirations from the social animal foraging strategy, there are still some obvious differences. The foraging frameworks adopted by the two algorithms are quite different: ACO utilizes the ant foraging behavior to perform optimization. Ants find food by laying down pheromone trails and collectively establishing positive feedbacks which bias the later path selection, while spiders sense the vibration propagated by the spider web to locate the prey. Another difference is the presentation of feasible solutions. In SSA we use the positions on the spider web to represent feasible solutions. Similar representations have also been widely adopted in the swarm intelligence algorithm. Meanwhile, ACO uses the path between the ant hive and food sources to represent solutions to the optimization problems. Additionally, ACO was originally designed to solve combinatorial problems. Although in recent years there are ACO-variant algorithms designed mainly to solve continuous problems [27], the performance is not as good as the performance of the original ACO in solving combinatorial problems like the Traveling Salesman Problem. There are also information propagation and searching pattern differences between SSA and ACO as described above.

There are also some other swarm intelligence algorithms proposed to solve continuous problems, and SSA has some unique characteristics. In most swarm intelligence algorithms, e.g., ABC and GSO, the populations are structured into different types. Different types of individuals perform different jobs and the whole population cooperates to search the solution space. However in SSA, all individuals (spiders) are equal. Each performs all the tasks that would be executed by multiple types of the populations in other algorithms. If we put SSA into the conventional framework, it has the feature that the different types of individuals can transform into other types very smoothly and without the guidance of the user, which may potentially contribute to the performance improvement.

As to social spider optimization (SSO) [25], differences lie in all aspects of the algorithm design. A most important difference is that in SSO the spiders are classified by gender. Male and female spiders have different searching operations. However, the spiders in SSA share the same searching operation, significantly reducing the effort in implementation. SSA also incorporates the information propagation model into its algorithm design, and thus the social spider population in SSA fits the IS model. Besides, SSA imitates the foraging behavior of social spiders, while SSO imitates the mating behavior of social spiders. The differences in algorithm implementation are more patent. In SSO there are three spider movement operators executed first in parallel and then in sequence. The moving pattern of the third operator highly depends on the first two operators. This design may potentially increase the difficulty of analyzing the search behavior of the algorithm. In SSA we implement one random move operator, which combines both exploration and exploitation behaviors in one move. In our design, the search behavior is controlled by the parameters, thus providing a clear view on the search behavior of the algorithm. The impact of different parameters on the optimization performance of SSA is further illustrated in Section 3.5.

Although EAs, like GA and ES, are also population-based algorithms, and inevitably share some similarities with the population-based SSA, they are quite different general-purpose metaheuristics. They are inspired by completely different biological disciplines. EAs usually employ different recombination and decomposition operators to manipulate the solutions, which imitate the regeneration of an organism.

As stated above, although we still do not know the exact impact of information loss on the optimization process, this feature of SSA may contribute to the optimum search in some complex multimodal optimization problems. The uniform structure of the population is another potential advantage of SSA. In addition, the unique searching pattern and its underlying social animal foraging strategy as well as the IS foraging model contribute to the overall performance of SSA.

3.5. Adjusting SSA parameters

Choosing proper parameters of SSA for numerical and real-world optimization problems can be time-consuming. The trial-and-error scheme, or a parameter sweep test, may reveal the best achievable performance over the parameter space at the expense of high computational cost. In real-world optimization problems, evaluating the fitness function may take a long time, much longer than evaluating our benchmark functions, and one evaluation may take several seconds or even minutes to finish, rendering trial-and-error schemes impractical for parameter tuning. As alternatives, researchers have proposed some schemes to replace the trial-and-error parameter selection scheme. These schemes can generally be classified into three groups [28]:

- *Fixed parameter schemes* select a parameter combination before the simulation using empirical or theoretical knowledge of the characteristics of the parameters. This combination remains constant throughout the whole search [29,30].
- *Deterministic parameter schemes* use some pre-defined rules to change the parameter values throughout the search [29,31].
- *Adaptive parameter schemes* change the parameter values by adaptively learning the impact of changing parameters on the searching performance throughout the search [32]. Some

schemes encode the parameters into the solution and evolve the parameters together with the population [33].

In this paper we use the fixed parameter scheme to test the performance of SSA compared with other algorithms. We also use this scheme to perform a preliminary parameter sensitivity analysis in order to deduce some rules of thumb on choosing parameters that can consistently lead to satisfactory results on a wide range of functions with different characteristics. This test can also discover some of the features of the parameters when solving different kinds of optimization problems. We carry out extensive simulations on our benchmark functions, which cover a wide range of optimization problems. Thus, the derived rules of thumb can be expected to give generally good performance on unknown problems.

In SSA we employ three user-controlled parameters to guide the searching behavior, namely,

- r_a: This parameter defines the rate of vibration attenuation when propagating over the spider web.
- p_c: This parameter controls the probability of the spiders changing their dimension mask in the random walk step.
- p_m: This parameter defines the probability of each value in a dimension mask to be one.

In this section, we employ five 10-dimensional benchmark functions to investigate the impact of these parameters on the performance of SSA. These functions are the Sphere, Schwefel 2.22, Rastrigin, Ackley, and Griewank functions and the detailed definition of these functions can be found in Section 4. All benchmark functions are not shifted. The value of r_a is selected from the set $\{\frac{1}{10}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{1}{2}, 1, 2, 3, 4, 5, 10\}$, and the values of p_c and p_m are both 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}. So for each function/parameter pair there are 11 data points for analysis. The stopping criteria is set to 100 000 evaluations and each function is tested for 20 times. The population size is set to 10 in accordance with our later simulation. The mean results are plotted in Fig. 1 with dots, and the second-order polynomial regression curve for each function is also plotted for demonstration. As the Sphere and Schwefel 2.22 functions are uni-modal functions while the other three are multi-modal functions, we can obtain some interesting observations from the mean results and the regression curve.

From Fig. 1 we can see SSA is very robust in solving unimodal problems when the three parameters change. All six sub-figures indicate outstanding performance in terms of fitness value. A general conclusion is that the value of p_c shall not be set to a very large value, e.g. 0.99.

When considering the multimodal problems, we can observe some obvious impact of the parameters on the performance. In terms of the vibration attenuation rate r_a, both the mean results and the regression curves favor relatively small values. A generally preferred value is one, while random values selected from (0, 3] shall also be able to yield good results.

While r_a presents relatively stable results in the multimodal problem tests, p_c has different impacts on different problems. Rastrigin function favors a small p_c while Rosenbrock function prefers a medium-large one. It seems that most p_c values can generate good results on Ackley function. So the selection of p_c highly depends on the nature of the optimization problems to be solved.

As to p_m, the simulation results reveal a similar tendency with r_a that the multimodal problems seem to prefer a relatively small p_m somewhere near 0.1. So we adopt the parameter combination $r_a = 1$, $p_c = 0.7$, and $p_m = 0.1$ in all of our later simulations. Please note that this parameter combination is not guaranteed to be the best one for solving all optimization problems, and parameter tuning is essential to address unfamiliar problems.

Please note that the parameter sensitivity analysis in this paper is a preliminary one. In this test only one parameter is tested while the remaining two is set unchanged, i.e., $r_a = 1$, $p_c = 0.7$, and $p_m = 0.1$. Although it is not guaranteed that this set of parameters works best for the benchmark functions, it is one that yields outstanding performance, which will be demonstrated in Section 4. A complete parameter sensitivity analysis is one of the future research topics of SSA.

4. Benchmark problems and evaluation method

In order to benchmark the performance of SSA, we conduct simulations on 25 different benchmark functions. These benchmark functions are all the base functions from the latest competition on real-parameter single objective optimization problems at CEC 2013 [34] and CEC 2014 [35]. The benchmark functions can be classified into four groups:

- Group I: f_1–f_5 are unimodal functions.
- Group II: f_6–f_{15} are multimodal functions.
- Group III: f_{16}–f_{20} are rotated multimodal functions whose base functions belong to Group II functions.
- Group IV: f_{21}–f_{25} are hybrid multimodal functions whose base functions belong to Group I–III functions.

The benchmark functions are listed in Table 1. All benchmark functions, except f_{13} Schwefel's Problem 2.26, are shifted minimization problems and the search ranges are scaled to $[-100, 100]^n$, where n is the dimension of the problem. Group I functions are used to test the fast-converging performance of SSA. Group II functions all have a large number of local minima points, and can be used to test the ability of SSA to jump out of local optima and avoid premature convergence. Group III functions are more complex than other functions and can push the searching capability of SSA to a limit. Group IV functions are employed to test the optimization performance of handling problems consisting of different subcomponents with different properties. The detailed implementation of Group IV functions, i.e. hybrid multimodal functions, can be found in [35]. In Table 1 we only list the subcomponents of the hybrid functions.

All benchmark functions locate their global minimum values at zero, and the fitness values smaller than 10^{-8} are considered as 10^{-8} as required in [34,35]. We test the benchmark functions in 10, 30, and 50 dimensions to draw empirical conclusion on the scalability of SSA, and each function is tested for 51 runs [35]. In each run, we use a maximum number of $10^4 \times n$ function evaluations as the termination criteria. In order to meet the requirement set by [35], we use one fixed combination of parameters for SSA in the simulation of all groups of functions. The population size is n, and other parameters are set according to the analysis in Section 3.5.

To evaluate the performance of SSA, we compare the simulation results with the state-of-the-art algorithms in solving real-parameter optimization problems, including the variances of Co-variance Matrix Adaptation Evolution Strategies (CMA-ES) [36], adaptive differential evolution (DE) algorithms [37], and a global and local real-coded genetic algorithm (GL-25) [38]. CMA-ES and DE variants are arguably the most successful optimization algorithms current in use [39]. In the latest Competition on Real-Parameter Single Objective Optimization Problems at CEC 2013, their variant algorithms possess nine positions in the top ten best performing algorithms. For DE variants, we select JADE [40] and SaDE [32] for comparison due to their excellent performance demonstrated in the CEC 2013 competition. All source codes are obtained from the original author. We make some minor changes to adapt them to our benchmark functions, but the main body and

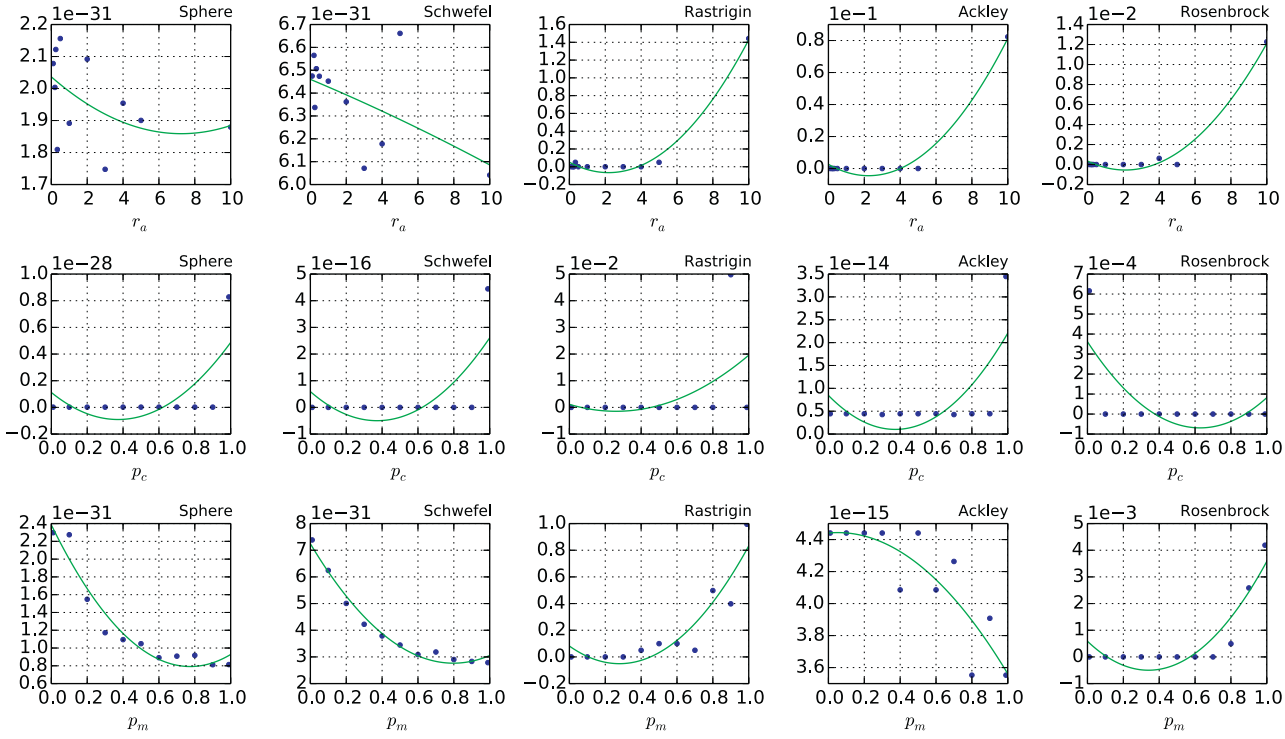

Fig. 1. Parameter test results on r_a, p_c, and p_m.

logic of the algorithms are untouched. The stopping criteria for all the compared algorithms are set to $10^4 \times n$ function evaluations, and the parameters of these algorithms are set according to the recommendation made in the corresponding literature, i.e., CMA-ES in [41], JADE in [40], SaDE in [32], and GL-25 in [38].

Besides these state-of-the-art algorithms, we also performed simulations with other famous algorithms, namely, Real-Coded Genetic Algorithm [42], Adaptive Particle Swarm Optimization [43], Artificial Bee Colony Optimization [18], Firefly Algorithm [22], Cuckoo Search [44], and Group Search Optimizer [45]. As their overall performance in terms of best fitness values achieved and convergence speed is not comparable to SSA and the other four algorithms, the detailed simulation results will not be presented in this paper.

5. Numerical experiments and results

In this section we present the simulation results of SSA on the benchmark functions identified in Section 4. We perform comparison among SSA and other algorithms and give statistical analysis on the simulation results.

5.1. Experimental comparison with other state-of-the-art algorithms

We first conduct a series of simulations on the 30-dimension optimization problems using SSA and other state-of-the-art algorithms. The simulation results are plotted in Fig. 2 and Table 2 reports the means and standard deviations of the optimal fitness values achieved, with the best mean result shaded. Besides, we also perform a series of Wilcoxon rank sum tests on the null hypothesis that SSA performs similarly with other algorithms when solving each benchmark function. The statistical test result at 95% significance level is presented next to the standard deviation of the corresponding algorithm, where a $\ominus$ indicates that SSA performs significantly better than the tested algorithm on the specified function, a $\oplus$ indicates that SSA performs not as good as the tested algorithm, and a $\odot$ means that the Wilcoxon rank sum test cannot distinguish between the simulation results of SSA and the tested algorithm. The counts of the benchmark functions that fall in these situations are shown at the bottom of the table.

From Table 2, we observe that:

- SSA generally outperforms all compared algorithms in terms of the statistical test. Among all 25 functions, SSA generates better simulation results in 16, 12, 14, and 13 functions compared with CMA-ES, JADE, SaDE, and GL-25, respectively. If we take those functions with similar results, the advantage is more obvious: SSA performs no worse than CMA-ES, JADE, SaDE, and GL-25 in 22, 19, 24, and 22 functions, respectively.
- In the first group of benchmark functions, all compared algorithms can obtain the global optimum values of f_1–f_4 in all runs, which means that the final result test cannot reveal the best-performing algorithms. We shall further employ the convergence test to analyze the performance of compared algorithms.
- The performance of SSA in solving Group II multimodal optimization problems is superior, and it generates 7 best mean results out of the total 10. The numbers for CMA-ES, JADE, SaDE, and GL-25 are 1, 4, 1, and 4, respectively. Besides, the mean results of the three functions where SSA is not the best performing one are still very competitive and comparable to the best results.
- SSA is not as competitive in solving rotated multimodal functions as it does in unimodal and multimodal functions. However, a careful investigation on the mean results shows that the performance of SSA is still comparable to all compared algorithm. The reason of this phenomenon may be that during the searching process of SSA, no correlation matrix or differential vectors are employed to assist the blind search in the solution space as in CMA-ES and JADE. However, this disadvantage can be overcome by employing these mentioned schemes into SSA or via hybrid algorithms. This is a potential future research direction.

Table 1
Benchmark functions.

Function	Transformation[a]	Name
$f_1(\boldsymbol{z}) = \sum_{i=1}^{n} z_i^2$	$\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{o}$	Sphere Function
$f_2(\boldsymbol{z}) = \sum_{i=1}^{n} \lvert z_i \rvert + \prod_{i=1}^{n} \lvert z_i \rvert$	$\boldsymbol{z} = (\boldsymbol{x} - \boldsymbol{o}) \times 10/100$	Schwefel's Problem 2.22
$f_3(\boldsymbol{z}) = z_1^2 + 10^6 \sum_{i=2}^{n} z_i^2$	$\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{o}$	Cigar Function
$f_4(\boldsymbol{z}) = 10^6 z_1^2 + \sum_{i=2}^{n} z_i^2$	$\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{o}$	Discus Function
$f_5(\boldsymbol{z}) = \sum_{i=1}^{n} i x_i^4 + rand()$[b]	$\boldsymbol{z} = (\boldsymbol{x} - \boldsymbol{o}) \times 1.28/100$	Quadratic Function with Noise
$f_6(\boldsymbol{z}) = \sum_{i=1}^{n} (z_i^2 - 10\cos(2\pi z_i) + 10)$	$\boldsymbol{z} = (\boldsymbol{x} - \boldsymbol{o}) \times 5.12/100$	Rastrigin Function
$f_7(\boldsymbol{z}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} z_i^2}\right) - \exp\left[\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi z_i)\right] + 20 + e$	$\boldsymbol{z} = (\boldsymbol{x} - \boldsymbol{o}) \times 32/100$	Ackley Function
$f_8(\boldsymbol{z}) = \frac{1}{4000}\sum_{i=1}^{n} z_i^2 - \prod_{i=1}^{n}\cos\left(\frac{z_i}{\sqrt{i}}\right) + 1$	$\boldsymbol{z} = (\boldsymbol{x} - \boldsymbol{o}) \times 600/100$	Griewank Function
$f_9(\boldsymbol{z}) = \sum_{i=1}^{n-1} (100(z_{i+1} - z_i^2)^2 + (z_i - 1)^2)$	$\boldsymbol{z} = (\boldsymbol{x} - \boldsymbol{o}) \times 30/100$	Rosenbrock Function
$f_{10}(\boldsymbol{z}) = sin^2(\pi y_1) + \sum_{i=1}^{n-1}[(y_i - 1)^2(1 + 10(\sin^2 y_{i+1}))] + (y_n - 1)^2(1 + \sin^2(2\pi y_n)),\ y_i = 1 + \frac{1}{4}(z_i + 1)$	$\boldsymbol{z} = (\boldsymbol{x} - \boldsymbol{o}) \times 50/100$	Levy Function
$f_{11}(\boldsymbol{z}) = \frac{1}{10}[\sin^2(3\pi z_1) + \sum_{i=1}^{n-1}(z_i - 1)^2(1 + \sin^2(3\pi z_{i+1})) + (z_n - 1)^2(1 + \sin^2(2\pi z_n))] + \sum_{i=1}^{n} u(z_i, 5, 100, 4)$ $u(z_i, a, k, m) = \begin{cases} k(z_i - a)^m & for z_i > a \\ 0 & for -a \le z_i \le a \\ k(-z_i - a)^m & for z_i < -a \end{cases}$	$\boldsymbol{z} = (\boldsymbol{x} - \boldsymbol{o}) \times 50/100$	Penalized Function
$f_{12}(\boldsymbol{z}) = g(z_1, z_2) + g(z_2, z_3) + \cdots + g(z_{n-1}, z_n) + g(z_n, z_1)$ $g(x, y) = 0.5 + \frac{(\sin^2(\sqrt{x^2 + y^2}) - 0.5)}{(1 + 0.001(x^2 + y^2))^2}$	$\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{o}$	Schaffer's Function F6
$f_{13}(\boldsymbol{z}) = 418.9828872724338 \times n - \sum_{i=1}^{n}(z_i \sin\sqrt{\lvert z_i \rvert})$	$\boldsymbol{z} = \boldsymbol{x} \times 500/100$	Schwefel's Problem 2.26
$f_{14}(\boldsymbol{z}) = [\frac{1}{n-1}\sum_{i=1}^{n-1}(\sqrt{y_i} + \sin(50y_i^{0.2})\sqrt{y_i})]^2,\ y_i = \sqrt{z_i^2 + z_{i+1}^2}$	$\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{o}$	Schaffer's Function F7
$f_{15}(\boldsymbol{z}) = \min(\sum_{i=1}^{n}(z_i - \mu_1)^2, d \times n + s \times \sum_{i=1}^{n}(z_i - \mu_2)^2) + 10\sum_{i=1}^{n}(1 - \cos[2\pi(z_i - \mu_1)])$ $s = 1 - \frac{1}{2\sqrt{n} - 8.2},\ \mu_1 = 2.5,\ \mu_2 = -\sqrt{\frac{\mu_1^2 - 1}{s}}$	$\boldsymbol{z} = (\boldsymbol{x} - \boldsymbol{o}) \times 10/100$	Lunacek Function
$f_{16}(\boldsymbol{z}) = f_8(\boldsymbol{Mz})$	$\boldsymbol{z} = (\boldsymbol{x} - \boldsymbol{o}) \times 600/100$	Rotated Griewank Function
$f_{17}(\boldsymbol{z}) = f_9(\boldsymbol{Mz})$	$\boldsymbol{z} = (\boldsymbol{x} - \boldsymbol{o}) \times 30/100$	Rotated Rosenbrock Function
$f_{18}(\boldsymbol{z}) = f_{11}(\boldsymbol{Mz})$	$\boldsymbol{z} = (\boldsymbol{x} - \boldsymbol{o}) \times 50/100$	Rotated Penalized Function
$f_{19}(\boldsymbol{z}) = f_{12}(\boldsymbol{Mz})$	$\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{o}$	Rotated Schaffer's Function F6
$f_{20}(\boldsymbol{z}) = f_{15}(\boldsymbol{Mz})$	$\boldsymbol{z} = (\boldsymbol{x} - \boldsymbol{o}) \times 10/100$	Rotated Lunacek Function
$f_{21}(\boldsymbol{z}) = f_1(\boldsymbol{z_1}) + f_6(\boldsymbol{z_2}) + f_{13}(\boldsymbol{z_3})$	$\boldsymbol{z} = [\boldsymbol{z_1}, \boldsymbol{z_2}, \boldsymbol{z_3}]$	Hybrid Function 1
$f_{22}(\boldsymbol{z}) = f_6(\boldsymbol{z_1}) + f_8(\boldsymbol{z_2}) + f_9(\boldsymbol{z_3})$	$\boldsymbol{z} = [\boldsymbol{z_1}, \boldsymbol{z_2}, \boldsymbol{z_3}]$	Hybrid Function 2
$f_{23}(\boldsymbol{z}) = f_3(\boldsymbol{z_1}) + f_7(\boldsymbol{z_2}) + f_9(\boldsymbol{z_3}) + f_{11}(\boldsymbol{z_4})$	$\boldsymbol{z} = [\boldsymbol{z_1}, \boldsymbol{z_2}, \boldsymbol{z_3}, \boldsymbol{z_4}]$	Hybrid Function 3
$f_{24}(\boldsymbol{z}) = f_6(\boldsymbol{z_1}) + f_7(\boldsymbol{z_2}) + f_8(\boldsymbol{z_3}) + f_9(\boldsymbol{z_4}) + f_{13}(\boldsymbol{z_5})$	$\boldsymbol{z} = [\boldsymbol{z_1}, \boldsymbol{z_2}, \boldsymbol{z_3}, \boldsymbol{z_4}, \boldsymbol{z_5}]$	Hybrid Function 4
$f_{25}(\boldsymbol{z}) = f_1(\boldsymbol{z_1}) + f_7(\boldsymbol{z_2}) + f_{10}(\boldsymbol{z_3}) + f_{13}(\boldsymbol{z_4}) + f_{15}(\boldsymbol{z_5})$	$\boldsymbol{z} = [\boldsymbol{z_1}, \boldsymbol{z_2}, \boldsymbol{z_3}, \boldsymbol{z_4}, \boldsymbol{z_5}]$	Hybrid Function 5

[a] $\boldsymbol{o}$ is a shifting vector and $\boldsymbol{M}$ is a transformation matrix. $\boldsymbol{o}$ and $\boldsymbol{M}$ can be obtained from [35].
[b] $rand()$ is a random number uniformly generated in (0, 1).

- SSA is very powerful at solving hybrid functions where different dimensions of the objective functions can be un-related and there is no additional information like correlation matrix available. In Group IV tests, SSA achieved four out of the total five best mean results.

5.2. Scalability test

In addition to the 30-dimension benchmark function tests, we also performed a series of simulations on both 10- and 50-dimension benchmarks to test the scalability of SSA. To make a thorough comparison, we also employed the compared algorithms in this test. The simulation results are presented in Tables 3 and 4, using the same format and symbols as in Table 2. From the results we have the following observations:

- The advantage of SSA over compared algorithms are confirmed. SSA achieved all the best mean results in 10-D Group II tests and 50-D Group IV tests, and have satisfactory performance compared with other algorithms in all other groups.
- This advantage is also supported by the statistical test. SSA can generate better results than CMA-ES, JADE, SaDE, and GL-25 in 18, 15, 19, and 8 10-D functions, respectively. The corresponding numbers for 50-D functions are 17, 9, 12, and 15, respectively. The statistical results will favor SSA more if we also take those tests that have similar performance into account.
- From the simulation results we can see GL-25 performs very well in 10-D hybrid functions, JADE performs very well in 50-D multimodal functions, when compared with the remaining three algorithms. However, SSA can always outperform them in these tests, which indicate the superior scalability of SSA.

5.3. Convergence test

As stated in Section 5.1, the final result comparison cannot completely describe the searching performance of an algorithm. So we further conduct a convergence test on the five compared algorithms on each 30-D benchmark function. We employ the raw simulation data generated in Section 5.1. As each function is tested for 51 runs for each algorithm, we select the convergence data of the run which generates the median final result. The convergence data of the five compared algorithms are plotted in Fig. 3. The x-axis is the function evaluations consumed, and the y-axis is the best-so-far fitness values found. The convergence plots lead to the following observations:

- The convergence speed of SSA in solving unimodal optimization problems is not as fast as CMA-ES, JADE, and SaDE. This is because

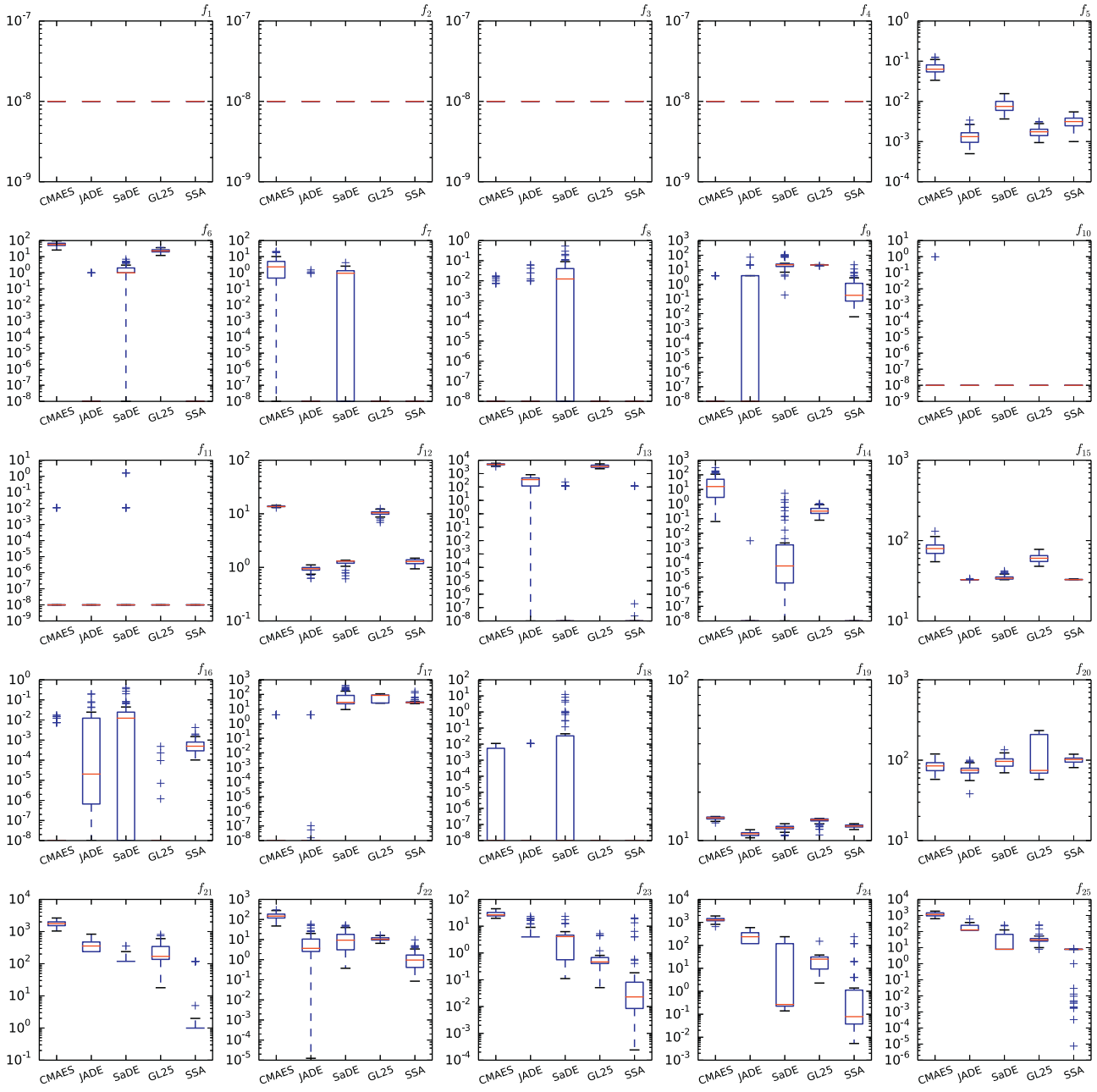

Fig. 2. Box plot of raw simulation results.

SSA performs exploitation and exploration simultaneously during the random walk process, and the former contributes to a fast convergence speed but the latter obstructs the population from moving into a small region in the search space. However, the degree of exploration in the random walk process can be effectively controlled by the algorithm parameters as revealed in Section 3.5. So it is highly possible that SSA can also achieve a comparable convergence speed with other algorithms if we adopt a set of suitable parameters designed for solving unimodal optimization problems.

- When solving multi-modal optimization problems, SSA generally converges as fast as or even faster than the compared algorithms. This phenomenon can be clearly observed in the Group II tests, where SSA mostly generates a similar convergence curve with others.
- The advantage of combining exploitation and exploration in one searching process is revealed in the convergence plot. Take f_6 as an example, almost all algorithms (except JADE) are trapped in local optima shortly after the start of searching. However, SSA managed to jump out of the optimum and successfully found the global optimum by the end of searching. The manifestation of this searching characteristic is that the algorithm convergences relatively slow at the beginning, but then very fast as the search continues. This phenomenon can also be observed in some other instances, e.g. f_{13}, f_{22}, and f_{24}.

5.4. Reliability test

Another test which can examine the reliability of stochastic algorithms is the success rate test and this test has been adopted in many previous work, e.g. [32,43]. In this part, we aim at further comparing and visualizing the performance of all compared algorithms on all tested benchmark functions by plotting the empirical cumulative distribution of success rates. The plots are presented

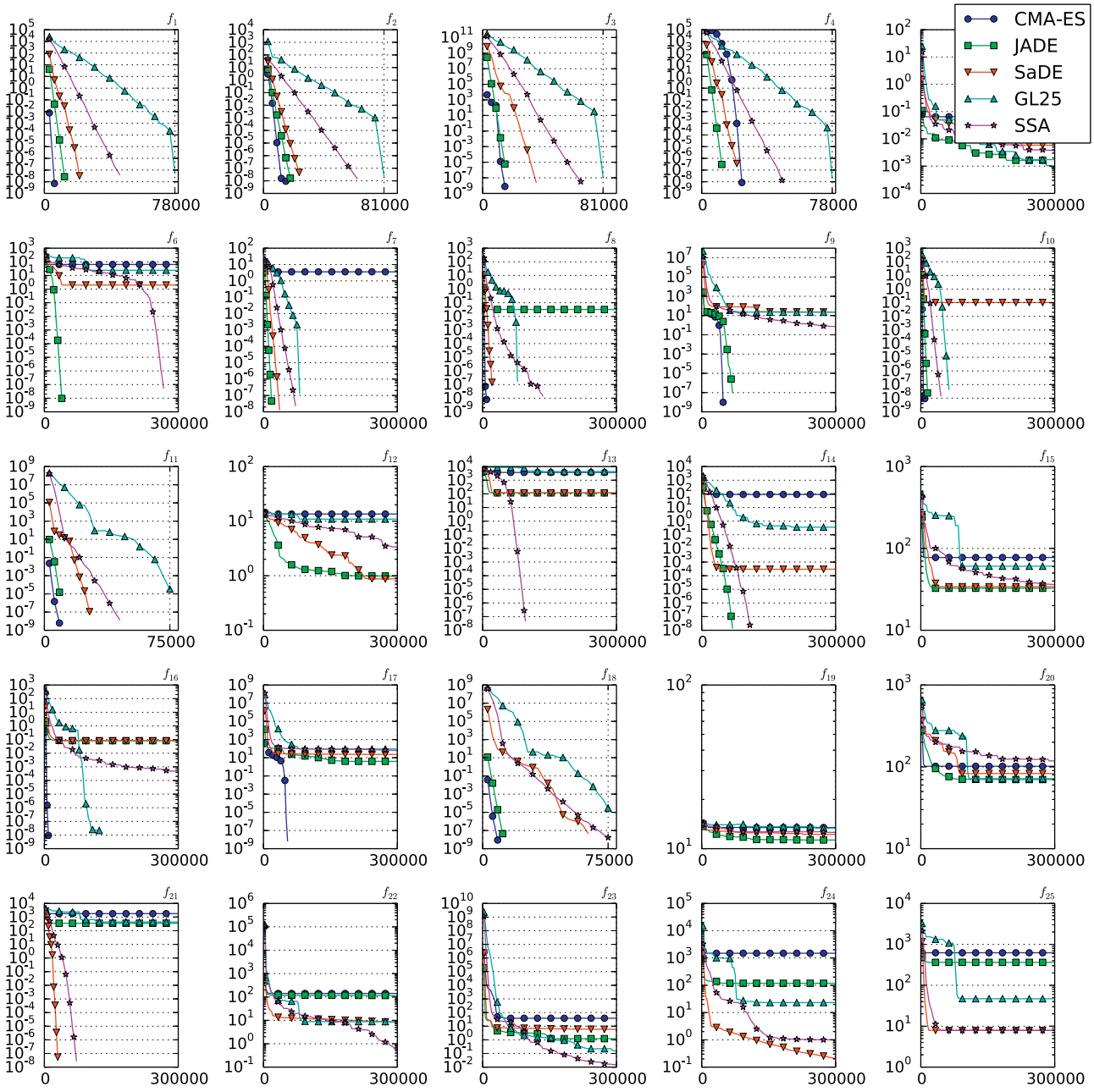

Fig. 3. Median convergence test results.

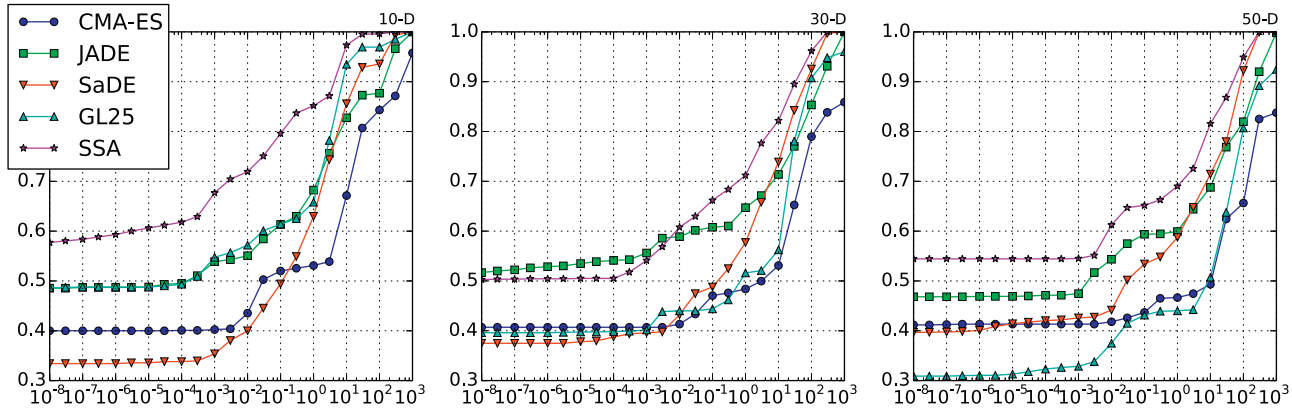

Fig. 4. Empirical cumulative distribution of success rate test results.

Table 2
Simulation results for 30-D problems.

Function	CMA-ES Mean±Std Dev	JADE Mean±Std Dev	SaDE Mean±Std Dev	GL25 Mean±Std Dev	SSA Mean±Std Dev
f_1	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_2	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_3	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_4	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_5	6.85E-02±2.21E-02⊖	1.39E-03±5.68E-04⊕	8.23E-03±3.22E-03⊖	1.77E-03±5.19E-04⊕	3.18E-03±9.99E-04
f_6	5.90E+01±1.56E+01⊖	7.80E-02±2.70E-01⊖	1.81E+00±1.59E+00⊖	2.36E+01±6.10E+00⊖	1.00E-08±0.00E+00
f_7	4.34E+00±5.77E+00⊖	8.86E-02±3.14E-01⊖	8.57E-01±8.68E-01⊖	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_8	1.84E-03±4.59E-03⊖	4.81E-03±1.39E-02⊖	4.40E-02±9.22E-02⊖	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_9	5.47E-01±1.39E+00⊕	3.72E+00±1.12E+01⊖	3.22E+01±2.99E+01⊖	2.12E+01±9.22E-01⊖	1.48E+00±3.62E+00
f_{10}	5.73E-02±2.31E-01⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_{11}	6.46E-04±2.61E-03⊙	1.00E-08±0.00E+00⊙	9.48E-02±3.79E-01⊖	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_{12}	1.37E+01±3.02E-01⊖	9.30E-01±1.10E-01⊕	1.22E+00±1.77E-01⊙	1.04E+01±1.07E+00⊖	1.27E+00±1.42E-01
f_{13}	4.89E+03±6.15E+02⊖	3.25E+02±1.86E+02⊖	2.79E+01±5.60E+01⊙	3.54E+03±7.98E+02⊖	1.16E+01±3.56E+01
f_{14}	4.21E+01±6.16E+01⊖	6.05E-05±4.32E-04⊙	2.46E-01±8.52E-01⊖	4.01E-01±2.44E-01⊖	1.00E-08±0.00E+00
f_{15}	8.02E+01±1.48E+01⊖	3.26E+01±1.57E-01⊕	3.48E+01±2.22E+00⊖	6.03E+01±7.02E+00⊖	3.28E+01±2.30E-01
f_{16}	2.32E-03±5.06E-03⊖	1.49E-02±4.05E-02⊖	3.76E-02±8.57E-02⊙	1.61E-05±7.55E-05⊕	6.76E-04±6.70E-04
f_{17}	8.60E-01±1.66E+00⊕	5.47E-01±1.39E+00⊕	8.07E+01±9.08E+01⊙	6.37E+01±3.32E+01⊖	3.43E+01±2.38E+01
f_{18}	2.80E-03±4.84E-03⊖	8.62E-04±2.98E-03⊖	6.94E-01±2.21E+00⊖	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_{19}	1.38E+01±2.84E-01⊖	1.10E+01±3.24E-01⊕	1.20E+01±4.23E-01⊕	1.33E+01±6.16E-01⊖	1.23E+01±2.42E-01
f_{20}	8.54E+01±1.57E+01⊕	7.52E+01±1.09E+01⊕	9.50E+01±1.37E+01⊕	1.17E+02±6.90E+01⊖	1.01E+02±8.78E+00
f_{21}	1.76E+03±3.47E+02⊖	3.53E+02±1.93E+02⊖	6.98E+01±9.82E+01⊖	2.41E+02±1.77E+02⊖	1.88E+01±4.34E+01
f_{22}	1.55E+02±4.92E+01⊖	9.81E+00±1.38E+01⊖	1.32E+01±1.33E+01⊖	1.11E+01±2.28E+00⊖	1.46E+00±1.69E+00
f_{23}	2.88E+01±6.89E+00⊖	3.90E+00±6.57E+00⊖	4.01E+00±4.38E+00⊖	7.43E-01±1.03E+00⊕	1.37E+00±4.32E+00
f_{24}	1.30E+03±2.60E+02⊖	2.58E+02±1.52E+02⊖	3.54E+01±5.93E+01⊖	2.38E+01±2.16E+01⊖	1.06E+01±3.98E+01
f_{25}	1.18E+03±3.07E+02⊖	1.89E+02±1.35E+02⊖	4.44E+01±6.77E+01⊖	4.06E+01±4.71E+01⊖	6.06E+00±3.35E+00
⊖	16	12	14	13	-
⊕	3	6	2	3	-
⊙	6	7	9	9	-

in Fig. 4, where the *x*-axis is different success thresholds, and the *y*-axis is the overall success rate. A simulation run is considered successful if and only if the best-found fitness value is smaller than or equal to the "successful threshold". The overall success rate is calculated by the number of successful runs under a specific success threshold divided by the total number of runs. Thus a larger overall success rate implies a more reliable algorithm. By comparing the distribution curves of different algorithms, we can have a general view on which algorithm is reliable at solving general optimization problems.

From the plots we have the following observations:

- SSA is generally more reliable than the compared algorithms. The advantage is very significant in 10-D and 50-D benchmark functions. While the reliability of SSA is very similar to JADE in small success threshold regions, SSA regains the lead in large threshold regions.
- The plot also demonstrates the convergence characteristic of SSA. SSA is able to find the global optimum whenever the algorithm is able to locate a relatively small region near it. This conclusion is

Table 3
Simulation results for 10-D problems.

Function	CMA-ES Mean±Std Dev	JADE Mean±Std Dev	SaDE Mean±Std Dev	GL25 Mean±Std Dev	SSA Mean±Std Dev
f_1	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_2	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_3	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_4	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_5	2.78E-02±1.87E-02⊖	4.58E-04±2.95E-04⊕	2.29E-03±1.98E-03⊖	4.70E-04±2.66E-04⊕	7.50E-04±3.48E-04
f_6	1.60E+01±7.81E+00⊖	2.34E+00±2.48E+00⊖	2.91E+00±2.74E+00⊖	4.61E+00±1.72E+00⊖	1.00E-08±0.00E+00
f_7	5.62E-01±2.90E+00⊖	4.50E-01±8.48E-01⊖	9.14E-01±1.16E+00⊖	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_8	9.95E-03±8.75E-03⊖	4.06E-02±6.97E-02⊖	1.15E-01±1.94E-01⊖	2.41E-03±6.32E-03⊖	1.00E-08±0.00E+00
f_9	3.13E-01±1.08E+00⊖	5.47E-01±1.39E+00⊖	3.08E+00±7.04E+00⊖	2.34E+00±6.81E-01⊖	2.60E-01±7.78E-01
f_{10}	3.82E-02±1.91E-01⊙	4.88E-02±1.14E-01⊖	7.32E-02±3.96E-01⊖	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_{11}	1.72E-03±4.04E-03⊖	3.42E-03±1.50E-02⊖	1.26E-01±5.57E-01⊖	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_{12}	4.24E+00±2.78E-01⊖	2.83E-01±2.46E-01⊙	4.18E-01±1.11E-01⊖	1.16E+00±7.46E-01⊖	2.20E-01±8.60E-02
f_{13}	1.61E+03±3.69E+02⊖	3.19E+02±1.81E+02⊖	1.15E+02±1.18E+02⊖	2.67E+02±2.21E+02⊖	2.32E+00±1.66E+01
f_{14}	1.36E+01±2.91E+01⊖	3.88E-01±2.51E+00⊖	1.44E+00±3.49E+00⊖	3.17E-03±5.31E-03⊖	1.00E-08±0.00E+00
f_{15}	1.36E+01±7.35E+00⊖	6.38E+00±6.57E+00⊙	1.14E+01±6.65E+00⊖	1.19E+01±3.78E+00⊖	4.04E+00±4.96E+00
f_{16}	9.85E-03±9.45E-03⊙	2.63E-02±3.09E-02⊖	1.17E-01±1.06E-01⊖	2.82E-03±4.99E-03⊕	4.35E-03±5.12E-03
f_{17}	4.69E-01±1.30E+00⊕	9.38E-01±1.71E+00⊕	1.75E+01±3.36E+01⊖	4.91E+00±6.39E-01⊕	1.49E+01±3.74E+01
f_{18}	1.70E-03±4.51E-03⊖	1.29E-03±3.58E-03⊖	1.87E-02±3.82E-02⊖	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_{19}	4.21E+00±2.21E-01⊖	2.76E+00±5.53E-01⊕	3.26E+00±3.83E-01⊙	3.09E+00±4.48E-01⊙	3.13E+00±2.32E-01
f_{20}	1.52E+01±6.98E+00⊖	7.67E+00±5.05E+00⊙	1.54E+01±8.07E+00⊖	6.02E+00±3.21E+00⊙	6.41E+00±2.46E+00
f_{21}	7.07E+02±2.56E+02⊖	2.18E+02±1.52E+02⊖	6.46E+01±9.33E+01⊖	1.74E+00±3.21E+00⊕	2.34E+00±1.66E+01
f_{22}	4.56E+01±2.88E+01⊖	1.06E+01±1.74E+01⊖	5.71E+00±7.26E+00⊖	4.02E-01±4.75E-01⊖	6.95E-02±1.42E-01
f_{23}	2.13E+01±4.43E+00⊖	3.44E+00±7.53E+00⊖	3.31E+00±7.02E+00⊙	7.76E-03±4.06E-02⊕	4.70E-01±2.85E+00
f_{24}	4.07E+02±1.66E+02⊖	1.03E+02±8.79E+01⊖	3.05E+01±6.38E+01⊖	1.15E+00±1.34E+00⊕	4.72E+00±3.32E+01
f_{25}	3.22E+02±1.63E+02⊖	1.00E+02±1.05E+02⊖	4.88E+01±7.23E+01⊖	1.95E-01±4.46E-01⊕	2.32E+00±1.66E+01
⊖	18	15	19	8	-
⊕	1	3	0	7	-
⊙	6	7	6	10	-

Table 4
Simulation results for 50-D problems.

Function	CMA-ES Mean±Std Dev	JADE Mean±Std Dev	SaDE Mean±Std Dev	GL25 Mean±Std Dev	SSA Mean±Std Dev
f_1	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_2	1.32E-08±1.69E-08⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_3	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_4	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_5	7.88E+00±5.55E+01⊖	2.25E-03±8.17E-04⊕	1.65E-02±6.08E-03⊖	4.22E-03±1.45E-03⊕	6.26E-03±1.32E-03
f_6	1.86E+02±2.76E+02⊖	1.00E-08±0.00E+00⊙	1.31E+00±1.10E+00⊖	4.98E+01±1.18E+01⊖	1.00E-08±0.00E+00
f_7	1.38E+01±8.17E+00⊖	1.60E-01±3.78E-01⊖	9.60E-01±6.57E-01⊖	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_8	5.80E-04±2.01E-03⊖	8.13E-03±1.70E-02⊖	3.89E-02±6.64E-02⊖	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_9	1.56E-01±7.82E-01⊕	1.02E+00±1.75E+00⊕	7.36E+01±2.81E+01⊖	4.13E+01±1.25E+00⊖	4.34E+00±9.45E+00
f_{10}	3.35E+00±1.22E+01⊖	2.44E-03±1.22E-02⊙	3.66E-03±1.93E-02⊙	1.00E-08±0.00E+00⊙	1.00E-08±0.00E+00
f_{11}	8.62E-04±2.98E-03⊖	2.15E-04±1.54E-03⊙	2.82E-01±9.77E-01⊖	6.57E-02±1.09E-01⊖	1.00E-08±0.00E+00
f_{12}	2.39E+01±8.29E-01⊖	1.73E+00±1.96E-01⊕	4.84E+00±1.33E+00⊕	1.91E+01±1.53E+00⊖	6.30E+00±6.35E-01
f_{13}	8.45E+03±7.98E+02⊖	1.14E+02±1.18E+02⊖	4.64E+00±2.32E+01⊙	7.39E+03±8.76E+02⊖	1.00E-08±0.00E+00
f_{14}	1.90E+02±1.40E+02⊖	1.00E-08±0.00E+00⊙	2.96E-01±8.43E-01⊖	8.74E+00±3.56E+00⊖	1.00E-08±0.00E+00
f_{15}	1.53E+02±2.51E+01⊖	5.00E+01±6.38E-14⊕	5.23E+01±1.72E+00⊕	1.07E+02±1.11E+01⊖	5.71E+01±4.62E+00
f_{16}	5.80E-04±2.39E-03⊕	2.20E-02±2.63E-02⊙	2.49E-02±2.81E-02⊙	1.42E-02±1.85E-02⊙	1.31E-02±7.03E-03
f_{17}	5.47E-01±1.39E+00⊕	9.46E-01±1.71E+00⊕	6.22E+01±3.07E+01⊙	4.70E+01±1.02E+00⊖	4.46E+01±2.63E+00
f_{18}	1.08E-03±3.30E-03⊖	1.29E-03±3.58E-03⊖	5.45E+00±1.60E+01⊖	1.31E-02±2.86E-02⊖	1.00E-08±0.00E+00
f_{19}	2.41E+01±7.83E-01⊖	1.98E+01±3.91E-01⊕	2.17E+01±2.58E-01⊕	2.34E+01±2.50E-01⊖	2.19E+01±2.19E-01
f_{20}	1.59E+02±1.92E+01⊕	1.19E+02±1.03E+01⊕	1.82E+02±2.54E+01⊕	1.99E+02±1.25E+02⊕	2.49E+02±1.62E+01
f_{21}	3.29E+03±5.11E+02⊖	5.74E+02±2.50E+02⊖	3.96E+01±8.09E+01⊖	1.42E+03±4.17E+02⊖	1.89E+01±4.34E+01
f_{22}	2.33E+02±6.38E+01⊖	2.33E+01±1.84E+01⊖	3.24E+01±1.80E+01⊖	2.40E+01±4.10E+00⊖	3.75E+00±3.37E+00
f_{23}	2.87E+01±8.78E+00⊖	1.04E+01±1.12E+01⊖	1.20E+01±2.07E+01⊖	4.68E+00±7.69E-01⊖	5.71E-01±1.56E+00
f_{24}	2.16E+03±4.29E+02⊖	3.36E+02±1.88E+02⊖	5.93E+01±8.32E+01⊙	2.10E+02±1.62E+02⊖	1.42E+01±2.82E+01
f_{25}	1.95E+03±3.67E+02⊖	3.11E+02±1.80E+02⊖	3.02E+01±5.72E+01⊖	1.94E+02±1.56E+02⊖	6.97E+00±2.81E+01
⊖	17	9	12	15	-
⊕	4	7	4	2	-
⊙	4	9	9	8	-

drawn based on the observation that almost all runs that obtained a fitness value smaller than 10^{-4} in 30-D functions, and 10^{-3} in 50-D functions are able to converge to the global optimum point at 10^{-8}.

5.5. Computational complexity

Besides the previous tests, the computational complexity is also a major factor for evaluating the efficiency of an evolutionary computation algorithm. In this paper, we employ the method stated in [34,35] to analyze the computational complexity of the compared algorithms. We use f_6 in Table 1, which is the major component of the testing methodology suggested by [35], as the benchmark evaluation function, and the complexity analysis result is as follows. The complexity values of CMA-ES, JADE, SaDE, GL-25, and SSA are 40.47, 34.69, 75.72, 63.51, and 44.18, respectively. A smaller complexity value means that the algorithm is less complex, which leads to a relatively faster speed in execution under the same condition. From the results we can see that although SSA is slightly more complicated than JADE and CMA-ES, their complexities are comparable. In addition, these three algorithms are significantly less computationally complex than SaDE and GL-25.

5.6. Discussion

As stated by the No-Free-Lunch (NFL) Theorem [46], all meta-heuristics that search for extrema shall perform exactly the same when all possible objective functions are evaluated and averaged. It is further elaborated that it is theoretically impossible to have a best general-purpose universal optimization technique [47]. Superior performing algorithms are available if particular classes of problems are considered [47], or general but real-world ones [48].

However, the total number of possible problems is so huge that there is still much room to develop new algorithms. Though existing meta-heuristics have great success in solving many optimization problems, it is always worthwhile to develop new searching methodologies with superior performance in particular classes of problems [49]. This is the motivation for us to propose SSA for solving global numerical optimization problems.

6. Conclusion

In this paper we proposed a novel social spider algorithm to solve global optimization problems. This algorithm is based on the foraging behavior of social spiders and the information-sharing foraging strategy. SSA is conceptually simple and relatively easy to implement. SSA can tackle a wide range of different continuous optimization problems and has the potential to be employed to solve real-world problems.

In order to evaluate the performance of SSA, we adopted a set of 25 benchmark functions which cover a large variety of different optimization problem types. We compared SSA with the state-of-the-art optimization algorithms, namely, CMA-ES, JADE, SaDE, and GL-25. These algorithms have been employed to solve a large set of different benchmark optimization functions and real-world problems, and demonstrated outstanding performance. The results show that the performance of SSA is outstanding compared with the above listed algorithms in all three different groups of functions. This conclusion was supported by both the simulation results and the statistics of the simulation data.

Future research on SSA can be divided into three categories: scheme research, algorithm research, and real-world application. The random walk scheme in the current SSA may be further improved using advanced optimization techniques and hybrid algorithms with deterministic heuristics or local search algorithms. New schemes can also be applied in the searching process of SSA for performance improvement. In terms of algorithm research, SSA has the potential to be applied to solve combinatorial problems. We note that some other swarm intelligence algorithms like PSO and ABC originally designed to solve continuous optimization problems have been successfully modified to solve combinatorial problem [50,51]. Although SSA only has three parameters besides

the population size, it is still very interesting to develop adaptive or self-adaptive schemes for SSA to control the parameters and reduce the effort in tuning parameters. Last but not least, it would be interesting to identify real-world applications which can be addressed using SSA effectively and efficiently.

James J.Q. Yu received the B.Eng. degree in Electrical and Electronic Engineering from the University of Hong Kong, Pokfulam, Hong Kong, in 2011. He is now a Ph.D. candidate at the Department of Electrical and Electronic Engineering of the University of Hong Kong. His current research interests include optimization algorithm design and analysis, evolutionary computation and its application, data mining, wireless communications, and power system.

Victor O.K. Li received SB, SM, EE and ScD degrees in Electrical Engineering and Computer Science from MIT in 1977, 1979, 1980, and 1981, respectively. He is Chair Professor of Information Engineering and Head of the Department of Electrical and Electronic Engineering at the University of Hong Kong (HKU). He also served as Associate Dean of Engineering, and Managing Director of Versitech Ltd., the technology transfer and commercial arm of HKU, and on the board of China.com Ltd. He is now serving on the boards of Sunevision Holdings Ltd. and Anxin-China Holdings Ltd., listed on the Hong Kong Stock Exchange. Previously, he was Professor of Electrical Engineering at the University of Southern California (USC), Los Angeles, California, USA, and Director of the USC Communication Sciences Institute. Sought by government, industry, and academic organizations, he has lectured and consulted extensively around the world. He has received numerous awards, including the PRC Ministry of Education Changjiang Chair Professorship at Tsinghua University, the UK Royal Academy of Engineering Senior Visiting Fellowship in Communications, the Croucher Foundation Senior Research Fellowship, and the Order of the Bronze Bauhinia Star, Government of the Hong Kong Special Administrative Region, China. He is a Registered Professional Engineer and a Fellow of the Hong Kong Academy of Engineering Sciences, the IEEE, the IAE, and the HKIE.

Acknowledgment

The authors would like to thank the anonymous reviewer for useful and constructive comments.

References

[1] E.-G. Talbi, Metaheuristics: From Design to Implementation, Wiley, 2009.
[2] R. Mallipeddi, S. Mallipeddi, P.N. Suganthan, M.F. Tasgetiren, Differential evolution algorithm with ensemble of parameters and mutation strategies, Appl. Soft Comput. 11 (2011) 1679–1696.
[3] R.S. Parpinelli, H.S. Lopes, New inspirations in swarm intelligence: a survey, Int. J. Bio-Inspired Comput. 3 (1) (2011) 1–16.
[4] M. Yim, Y. Zhang, D. Duff, Modular robots, IEEE Spectr. 39 (2) (2002) 30–34.
[5] R. Foelix, Biology of Spiders, Oxford University Press, New York, NY, 1996.
[6] C.F. Schaber, S.N. Gorb, F.G. Barth, Force transformation in spider strain sensors: white light interferometry., J. R. Soc. Interface 9 (71) (2012) 1254–1264.
[7] P.K. Stoddard, V.L. Salazar, Energetic cost of communication, J. Exp. Biol. 214 (2011) 200–205.
[8] J. House, K. Landis, D. Umberson, Social relationships and health, Science 241 (4865) (1988) 540–545.
[9] C.W. Clark, M. Mangel, Foraging and flocking strategies: information in an uncertain environment, Am. Nat. 123 (5) (1984) 626–641.
[10] C. Barnard, R. Sibly, Producers and scroungers: a general model and its application to captive flocks of house sparrows, Anim. Behav. 29 (2) (1981) 543–550.
[11] G. Uetz, Foraging strategies of spiders, Trends Ecol. Evolut. 7 (5) (1992) 155–159.
[12] F. Fernndez Campn, Group foraging in the colonial spider *Parawixia bistriata* (Araneidae): effect of resource levels and prey size, Anim. Behav. 74 (5) (2007) 1551–1562.
[13] M. Dorigo, Optimization, learning and natural algorithms, Politecnico di Milano, Italie, 1990 (Ph.D. thesis).
[14] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proc. IEEE Int. Conf. Neural Networks, Perth, WA, 1995, pp. 1942–1948.
[15] T. Liao, D. Molina, T. Stutzle, M. Oca, M. Dorigo, An ACO algorithm benchmarked on the BBOB noiseless function testbed, in: Proc. 14th Int. Conf. GECCO, Philadelphia, USA, 2012, pp. 221–228.
[16] U. Kirchmaier, S. Hawe, K. Diepold, A swarm intelligence inspired algorithm for contour detection in images, Appl. Soft Comput. 13 (2013) 3118–3129.
[17] H.A. Abbass, MBO: marriage in honey bees optimization – a haplometrosis polygynous swarming approach, in: Proc. IEEE Congress on Evolutionary Computation (CEC), Seoul, Korea, 2001, pp. 207–214.
[18] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony, J. Glob. Optim. 39 (3) (2007) 459–471.
[19] S. Omkar, J. Senthilnath, R. Khandelwal, G.N. Naik, S. Gopalakrishnan, Artificial bee colony (ABC) for multi-objective design optimization of composite structures, Appl. Soft Comput. 11 (2011) 489–499.
[20] B. Akay, A study on particle swarm optimization and artificial bee colony algorithms for multilevel thresholding, Appl. Soft Comput. 13 (2013) 3066–3091.
[21] K. Krishnanand, D. Ghose, Detection of multiple source locations using a glowworm metaphor with applications to collective robotics, in: Proc. IEEE Swarm Intell. Symposium, Pasadena, CA, 2005, pp. 84–91.
[22] X.-S. Yang, Nature-inspired metaheuristic algorithms, in: Firefly Algorithm, Luniver Press, 2008, pp. 81–96 (Chapter 10).
[23] K.M. Passino, Biomimicry of bacterial foraging for distributed optimization and control, IEEE Control Syst. Mag. 22 (3) (2002) 52–67.
[24] M. Farhoodnea, A. Mohamed, H. Shareef, H. Zayandehroodi, Optimum placement of active power conditioner in distribution systems using improved discrete firefly algorithm for power quality enhancement, Appl. Soft Comput. 23 (2014) 249–258.
[25] E. Cuevas, M. Cienfuegos, D. Zaldvar, M. Prez-Cisneros, A swarm optimization algorithm inspired in the behavior of the social-spider, Expert Syst. Appl. 40 (16) (2013) 6374–6384.
[26] W. Chu, X. Gao, S. Sorooshian, Handling boundary constraints for particle swarm optimization in high-dimensional search space, Inf. Sci. 181 (20) (2011) 4569–4581.
[27] K. Socha, M. Dorigo, Ant colony optimization for continuous domains, Eur. J. Oper. Res. 185 (3) (2008) 1155–1173.
[28] A.K. Qin, X. Li, Differential evolution on the CEC-2013 single-objective continuous optimization testbed, Proc. IEEE Congress on Evolutionary Computation (CEC), Cancun, Mexico, 2013, pp. 1099–1106.
[29] A.Y.S. Lam, V.O.K. Li, J.J.Q. Yu, Real-coded chemical reaction optimization, IEEE Trans. Evol. Comput. 16 (3) (2012) 339–353.
[30] K. Price, R.M. Storn, J.A. Lampinen, Differential Evolution – A Practical Approach to Global Optimization, Springer, 2005.
[31] W.-N. Chen, J. Zhang, Y. Lin, N. Chen, Z.-H. Zhan, H.S.-H. Chung, Y. Li, Y.-H. Shi, Particle swarm optimization with an aging leader and challengers, IEEE Trans. Evol. Comput. 17 (2) (2013) 241–258.
[32] A. Qin, V. Huang, P. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE Trans. Evol. Comput. 13 (2) (2009) 398–417.
[33] J.A. Vrugt, B.A. Robinson, J.M. Hyman, Self-adaptive multimethod search for global optimization in real-parameter spaces, IEEE Trans. Evol. Comput. 13 (2) (2009) 243–259.
[34] J.J. Liang, B.-Y. Qu, P.N. Suganthan, A.G. Hernndez-Daz, Problem definitions and evaluation criteria for the CEC 2013 special session and competition on real-parameter optimization, Technical Report 201212, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Nanyang Technological University, Singapore, 2013.
[35] J.J. Liang, B.-Y. Qu, P.N. Suganthan, Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization, Technical Report 201311, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Nanyang Technological University, Singapore, 2014.
[36] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evol. Comput. 9 (2) (2001) 159–195.
[37] R. Storn, K. Price, Differential evolution: a simple and efficient heuristic for global optimization over continuous spaces, J. Glob. Optim. 11 (4) (1997) 341–359.
[38] C. Garcia-Martinez, M. Lozano, F. Herrera, D. Molina, A. Sanchez, Global and local real-coded genetic algorithms based on parent-centric crossover operators, Eur. J. Oper. Res. 185 (2008) 1088–1113.
[39] S. Das, P.N. Suganthan, Differential evolution: a survey of the sate-of-the-art, IEEE Trans. Evol. Comput. 15 (1) (2011) 4–31.
[40] J. Zhang, A.C. Sanderson, JADE: adaptive differential evolution with optional external archive, IEEE Trans. Evol. Comput. 13 (5) (2009) 945–958.
[41] I. Loshchilov, CMA-ES with restarts for solving CEC 2013 problems, in: Proc. IEEE Congress on Evolutionary Computation (CEC), Cancun, Mexico, 2013, pp. 369–376.
[42] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.
[43] Z.-H. Zhan, J. Zhang, Y. Li, H.S.-H. Chung, Adaptive particle swarm optimization, IEEE Trans. Evol. Comput. 39 (6) (2009) 1362–1381.
[44] X.-S. Yang, S. Deb, Engineering optimisation by cuckoo search, Int. J. Math. Modell. Numer. Optim. 1 (4) (2009) 330–343.
[45] S. He, Q.H. Wu, J.R. Saunders, Group search optimizer: an optimization algorithm inspired by animal searching behavior, IEEE Trans. Evol. Comput. 13 (5) (2009) 973–990.
[46] D. Wolpert, W. Macready, No free lunch theorems for optimization, IEEE Trans. Evol. Comput. 1 (1) (1997) 67–82.

[47] Y.C. Ho, D.L. Pepyne, Simple explanation of the no-free-lunch theorem and its implications, J. Optim. Theory Appl. 115 (3) (2002) 549–570.
[48] C. Garcia-Martinez, F.J. Rodriguez, M. Lozano, Arbitrary function optimisation with metaheuristics: no free lunch and real-world problems, Soft Comput. 16 (2012) 2115–2133.
[49] A.Y.S. Lam, V.O.K. Li, Chemical-reaction-inspired metaheuristic for optimization, IEEE Trans. Evol. Comput. 14 (3) (2010) 381–399.
[50] J. Kennedy, R.C. Eberhart, A discrete binary version of the particle swarm algorithm, in: Proc. IEEE Int. Conf. Syst. Man Cyber., Orlando, FL, 1997, pp. 4104–4108.
[51] Q.-K. Pan, M.F. Tasgetiren, P. Suganthan, T.J. Chua, A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem, Inf. Sci. 181 (12) (2011) 2455–2468.