An Optimization Algorithm Based on Brainstorming Process

Yuhui Shi, Xi'an Jiaotong-Liverpool University, China

ABSTRACT

In this paper, the human brainstorming process is modeled, based on which two versions of Brain Storm Optimization (BSO) algorithm are introduced. Simulation results show that both BSO algorithms perform reasonably well on ten benchmark functions, which validates the effectiveness and usefulness of the proposed BSO algorithms. Simulation results also show that one of the BSO algorithms, BSO-II, performs better than the other BSO algorithm, BSO-I, in general. Furthermore, average inter-cluster distance D_c and inter-cluster diversity D_e are defined, which can be used to measure and monitor the distribution of cluster centroids and information entropy of the population over iterations. Simulation results illustrate that further improvement could be achieved by taking advantage of information revealed by D_c and or D_e, which points at one direction for future research on BSO algorithms.

Keywords: *Algorithm, Brain Storm Optimization, Brainstorming Process, Diversity, Optimization*

INTRODUCTION

Many real-world applications can be represented as optimization problems of which algorithms are required to have the capability to search for optimum. Originally, these optimization problems were mathematically represented by continuous and differentiable functions so that algorithms such as hill-climbing algorithms can be designed and/or utilized to solve them. Traditionally, these hill-climbing like algorithms are single-point based algorithms such as gradient decent algorithms which move from the current point along the direction pointed by the negative of the gradient of the function at the current point. These hill-climbing algorithms can find solutions quickly for unimodal problems, but they have the problems of being sensitive to initial search point and being easily trapped into local optimum for nonlinear multimodal problems. Furthermore, these mathematical functions need to be continuous and differentiable, which instead greatly narrows the range of real-world problems that can be solved by hill-climbing algorithms. Recently, evolutionary algorithms have been designed and utilized to solve optimization problems. Different from traditional single-point based algorithms such as hill-climbing algorithms, each evolutionary algorithm is a population-based algorithm, which consists of a set of points (population of individuals). The population of individuals is expected to have high tendency to move towards better and better solution areas iteration over iteration through cooperation and/or competition among themselves. There are

DOI: 10.4018/jsir.2011100103

a lot of evolutionary algorithms out there in the literature. The most popular evolutionary algorithms are evolutionary programming (Fogel, 1962), genetic algorithm (Holland, 1975), evolution strategy (Rechenberg, 1973), and genetic programming (Koza, 1992), which were inspired by biological evolution. In evolutionary algorithms, population of individuals survives into the next iteration. Which individual has higher probability to survive is proportional to its fitness value according to some evaluation function. The survived individuals are then updated by utilizing evolutionary operators such as crossover operator and mutation operator, *etc*. In evolutionary programming and evolution strategy, only the mutation operation is employed, while in genetic algorithms and genetic programming, both the mutation operation and crossover operation are employed. The optimization problems to be optimized by evolutionary algorithms do not need to be mathematically represented as continuous and differentiable functions, they can be represented in any form. Only requirement for representing optimization problems is that each individual can be evaluated as a value called fitness value. Therefore, evolutionary algorithms can be applied to solve more general optimization problems, especially those that are very difficult, if not impossible, for traditional hill-climbing algorithms to solve.

Recently, another kind of algorithms, called swarm intelligence algorithms, is attracting more and more attentions from researchers. Swarm intelligence algorithms are usually nature-inspired optimization algorithms instead of evolution-inspired optimization algorithms such as evolutionary algorithms. Similar to evolutionary algorithms, a swarm intelligence algorithm is also a population-based optimization algorithm. Different from the evolutionary algorithms, each individual in a swarm intelligence algorithm represents a simple object such as ant, bird, fish, *etc*. So far, a lot of swarm intelligence algorithms have been proposed and studied. Among them are particle swarm optimization(PSO) (Eberhart & Shi, 2007; Shi & Eberhart, 1998), ant colony optimization algorithm(ACO) (Dorigo, Maniezzo, & Colorni, 1996), bacterial forging optimization algorithm(BFO) (Passino, 2010), firefly optimization algorithm (FFO) (Yang, 2008), bee colony optimization algorithm (BCO) (Tovey, 2004), artificial immune system (AIS) (de Castro & Von Zuben, 1999), fish school search optimization algorithm(FSO) (Bastos-Filho, De Lima Neto, Lins, Nascimento, & Lima, 2008), shuffled frog-leaping algorithm (SFL) (Eusuff & Lansey, 2006), intelligent water drops algorithm (IWD) (Shah-Hosseini, 2009), to just name a few.

In a swarm intelligence algorithm, an individual represents a simple object such as birds in PSO, ants in ACO, bacteria in BFO, *etc*. These simple objects cooperate and compete among themselves to have a high tendency to move toward better and better search areas. As a consequence, it is the collective behavior of all individuals that makes a swarm intelligence algorithm to be effective in problem optimization.

For example, in PSO, each particle (individual) is associated with a velocity. The velocity of each particle is dynamically updated according to its own historical best performance and its companions' historical best performance. All the particles in the PSO population fly through the solution space in the hope that particles will fly towards better and better search areas with high probability.

Mathematically, the updating process of the population of individuals over iterations can be looked as a mapping process from one population of individuals to another population of individuals from one iteration to the next iteration, which can be represented as $P_{t+1} = f(P_t)$, where P_t is the population of individuals at the iteration t, $f()$ is the mapping function. Different evolutionary algorithm or swarm intelligence algorithm has a different mapping function. Through the mapping function, we expect the population of individuals will update to better and better solutions over iterations. Therefore mapping functions should possess the property of convergence. For nonlinear and complicated problems, mapping functions more

like to move population of individuals toward local minima, which may not be good enough solutions to the optimization problems to be solved. A good mapping function should have not only the capability to converge, but also the capability to diverge when it gets trapped into local minima. As for evolutionary algorithms and swarm intelligence algorithms, they should have the capability to be in convergence or divergence state accordingly. A lot of researches have been done and reported with regards to this. For example, in particle swarm optimization algorithms, diversity has been preserved to keep the algorithm to have good search capability. Different diversity measurements have been defined and monitored (Shi & Eberhart, 2008, 2009). A better designed population-based algorithm should have a good balance of convergence and divergence.

In this paper, we will introduce a new optimization algorithm that is based on the collective behavior of human being, that is, the brainstorming process. It is natural to expect that an optimization algorithm based on human collective behavior could be a better optimization algorithm than existing swarm intelligence algorithms which are based on collective behavior of simple insects, because human beings are social animal and are the most intelligent animals in the world. The designed optimization algorithm will naturally have the capability of both convergence and divergence.

The remaining paper is organized as follows. The human brainstorming process is reviewed. The model of a brainstorming process is proposed and discussed. Two versions of novel optimization algorithms inspired by human brainstorming process are introduced and described, followed by experiments and result discussion on benchmark functions. Finally, conclusions are given.

BRAINSTORMING PROCESS

Brainstorming process has often been utilized for innovative problem solving. It can solve a lot of difficult problems which usually can't be solved by a single person. In a brainstorming process, a group of people with diverse background are gathered together to brainstorm. A facilitator will usually be involved to facilitate the brainstorming process but not directly involved in idea generation himself (or herself). The facilitator usually should have enough facilitation experience but have less knowledge about the problem to be solved so that generated ideas will have less, if not none, biases from the facilitator. The brainstorming process is used to generate many ideas as diverse as possible so that good solutions to solve the problem can be obtained from these ideas. The brainstorming process usually consists of several rounds of idea generation. In each round of idea generation, the brainstorming group is asked to come out a lot of ideas. At the end of each round of idea generation process, better ideas among them will be picked up and will serve as clues to generate ideas in the next round of idea generation process. In the brainstorming process, there is another group of persons that serve the purpose to pick up better ideas from the ideas generated in each round of idea generation process. Through the brainstorming process, hopefully great and un-expectable solution can occur from collective intelligence of human being, and the problem can usually be solved with high probability.

To help generate more diverse ideas, the Osborn's original four rules of idea generation in a brainstorming process (Osborn, 1963; Smith, 2002) should be obeyed. The four rules are listed in the Table 1. One major role of the facilitator is to facilitate the brainstorming group to obey the Osborn's four rules.

The four rules in Table 1 guide the idea generation in each round of idea generation during a brainstorming process. In order to keep the brainstorming group to be open-minded, there is no idea as good idea or bad idea, any idea is welcomed. For any idea generated during each round of idea generation process, there should be no judgment and/or criticism whether it is good idea or bad idea. Any judgment should be held back until the end of this round of idea generation process when better ideas

Table 1. Osborn's original rules for idea generation in a brainstorming process

1. Suspend Judgment
2. Anything Goes
3. Cross-fertilize (Piggyback)
4. Go for Quantity

Table 2. Steps in a brainstorming process

1. Get together a brainstorming group of people with as diverse background as possible;
2. Generate many ideas according to the rules in Table 1;
3. Have several, say 3 or 5, clients act as the owners of the problem to pick up several, say one from each owner, ideas as better ideas for solving the problem;
4. Use the ideas picked up in the Step 3 with higher probability than other ideas as clues, and generate more ideas according to the rules in Table 1;
5. Have the owners to pick up several better ideas generated as did in Step 3;
6. Randomly pick an object and use the functions and appearance of the object as clues, generate more ideas according to the rules in Table 1;
7. Have the owners to pick up several better ideas;
8. Hopefully a good enough solution can be obtained by considering the ideas generated.

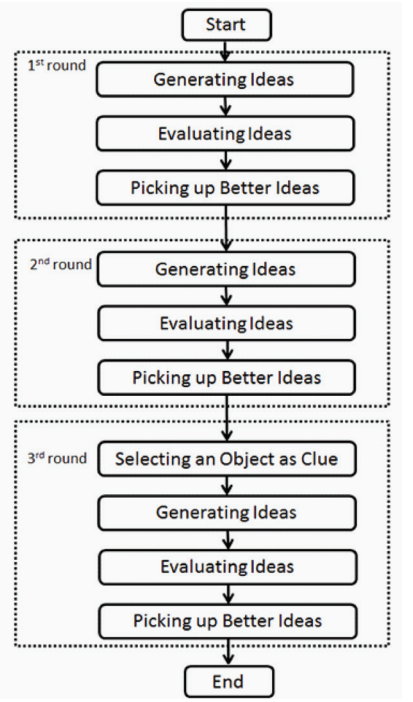
are picked up by problem owners. This is what the Rule 1 "Suspend Judgment" means. The Rule 2 "Anything Goes" means that any thought comes to your mind should be raised and recorded. Don't let any idea or thought pass by without sharing with other brainstorming group members. The Rule 3 "Piggyback" says any generated idea could and should serve as a clue to inspire the brainstorming group to come out more ideas. Ideas are not independently generated. They are related. The late generated ideas are inspired and dependent on the previously generated ideas. The Rule 4 "Go for quantity" says that we focus on generating as many ideas as possible. Hopefully quality of ideas will come out of quantity of idea naturally. Without generating large quantity of ideas, it is naive to believe that good quality ideas will come out.

The purpose to generate ideas according to rules in Table 1 is to keep the brainstorming group to be open-minded as much as possible so that they will generate ideas as diverse as possible. A brainstorming process generally follows the steps listed in Table 2 (Shi, 2011). After some time of brainstorming, the brainstorming group will become tired and narrow-minded, and therefore it becomes harder to come out new diverse ideas. The operation of picking up an object in Step 6 in Table 2 serves for the purpose to help brainstorming group to diverge from previously generated ideas therefore to avoid being trapped by the previously generated ideas. Picking up several good ideas from ideas generated so far is to cause the brainstorming group to pay more attention to the better ideas which the brainstorming group believes to be. The ideas picked-up works like point-attraction for the idea generation process while ideas generation works like point-expansion. Therefore, there are attraction and expansion embedded in the brainstorming process naturally.

MODELING BRAINSTORMING PROCESS

The procedure of a brainstorming process listed in Table 2 can be described by the flow chart shown in Figure 1. There are three rounds of idea generation involved in a brainstorming process in general. In each round of brainstorming process, there are several steps. For example, in the first round, there are idea generations, idea evaluations, and idea picking up. The idea evaluation step serves the purpose of finding

Figure 1. Flow chart of a brainstorming process



out better ideas. By idea evaluation, good ideas could be identified and picked up in the Picking up Better Ideas step, which simulates picking up good ideas by problem owners. The first round simulates Step 2 &3 in Table 2. The second round is the same as the first round which simulates Step 4 & 5 in Table 2. The third round is the same as the first two rounds except that one additional step Selecting an Object as Clue is added to simulate randomly picking up an object as clues in Step 6 in Table 2. Each step in a brainstorming process therefore can be modeled (and/or simulated) and put together as a model for the brainstorming process as shown in Figure 1, which will be further explained and modified in the following sub-sections.

Population

A solution to a problem with d variables to be optimized can be looked as a point in the d dimensional solution space. An idea can be considered as a potential solution, i.e., a point in the solution space. Therefore to find a good solution is equivalent to find a point or a solution in the solution space. A group of ideas can therefore be considered as a population of solutions or individuals in the solution space. If for every round of idea generation in the brainstorming process, a fixed number of n ideas will be generated before the problem owners pick up good ideas, then these n ideas can be considered as a population of individuals (or solutions) with population size being n in the solution space. Therefore, the human brainstorming process can be considered as generating a population of individuals iteratively three times as shown in Figure 1. One round of idea generation can be considered as one iteration of individual generations in population-based optimization algorithms such as particle swarm optimization algorithm. The difference between them is the way how new population of individuals is generated based on the current population of individuals.

Initialization

The Generating Ideas step in the very first round of idea generations can be considered as the population initialization in any population-based optimization algorithm. During the population initialization, to gather a group of people with as diverse background as possible can be considered as initializing the population of individuals randomly with uniform distribution over the dynamic range of the solution space. The whole population of individuals can be totally randomly generated or only portion of the population is randomly generated and the rest of the population of individuals will be generated by adding noise to the already randomly generated individuals. To preserve the initialized population to be diversified, usually *a priori* domain knowledge should not be utilized in the initialization process, unless when computation cost is the first priority, in which the domain knowledge should be utilized to initialize the population to find good solution quickly at the risk of premature convergence.

Clustering

Each round of idea generation generates enough ideas, but not necessary too many ideas because otherwise all the generated ideas will more like to diverge, and therefore will be far away from expected ideas which are close to expected solutions. To have diverse ideas is good to seek around all possible ideas to help find good potential solutions, but there should be a tradeoff between divergence and focus. We also need to pull the brainstorming group back to concentrate on generating ideas around some areas with high potential to speed up searching for good enough ideas. The problem owners in the brainstorming process serve this purpose. They are asked to pick good ideas from generated ideas. Because every problem owner has different expertise and knowledge, therefore the picked ideas will be different. They represent potential good ideas that have been generated so far. Next round of idea generation should better be conducted with focus on them. Certainly, it does not exclude idea generation by piggybacking other ideas, but with small probability. One way to simulate the idea picking up by problem owners is to use clustering algorithms. All the individuals (ideas) in the population are clustered into several clusters. The number of clusters corresponds to the number of problem owners. The cluster center of each cluster corresponds to the idea(s) picked up by a problem owner. The cluster center for each cluster can be the best performed individual within this cluster. It can also be the centroid of the cluster.

One possible clustering algorithm is the k-means clustering algorithm (MacQueen, 1967), which requires to know the number of clusters *k a priori*. The number *k* corresponds to the number of problem owners, that is, the number of problem owners is fixed. The self-organizing feature map (Kohonen & Honkela, 2007) is another kind of clustering algorithm, in which the number of clusters is unknown before running the algorithm. The number of clusters will be determined by the algorithm itself according to the distribution of individuals in the population. Other clustering algorithms (Xu & Wunsch, 2005) such as partitioning around medoids (Theodoridis & Koutroumbas, 2006), fuzzy c-means (FCM) (Nock & Nielsen, 2006), *etc.* can also be employed.

Individual Generation

For idea generations by piggyback, it is similar to randomly select one or several existing individuals (or ideas) and generate a new individual by adding noise to the selected individual(s). The purpose of doing this is to guarantee that new individuals (ideas) are generated by piggybacking existing individuals as diverse as possible. If a new idea (individual) x_{new} is generated by piggybacking one existing idea (individual) x_{old}, it can be written as

$$x_{new}^{i} = x_{old}^{i} + \xi(t) * random(t) \quad (1)$$

where $x_{new}{}^{i}$ and $x_{old}{}^{i}$ are the ith dimension of x_{new} and x_{old}, respectively; *random(t)* is a random

function; $\xi(t)$ is a coefficient that weights the contribution of random value to the new individual. The formula is similar to the mutation operation in evolutionary programming algorithm. The commonly utilized random function in mutation operation is the Gaussian function (Yao, Liu, & Lin, 1997). Other random functions that can be used are Cauchy function (Yao, Liu, & Lin, 1997), Lévy flights (Pavlyukevich, 2007), *etc*. Compared with Gaussian function, Cauchy function has a longer tail which makes it preferable if wider areas need to be explored (Yao, Liu, & Lin, 1997).

If a new idea (individual) x_{new} is generated by piggybacking two existing ideas (individuals) x_{old1} and x_{old2}, it can be written as

$$x_{new}^{i} = x_{old}^{i} + \xi(t) * random(t) \quad (2a)$$

$$x_{old}^{i} = w_1 * x_{old1}^{i} + w_2 * x_{old2}^{i} \quad (2b)$$

where x_{old}^{i} is the weighted summation of the ith dimension of x_{old1} and x_{old2}; w_1 and w_2 are two coefficients to weight the contribution of two existing individuals. The formula simulates generating new idea by piggybacking two existing ideas. Certainly, a new idea can also be generated by piggybacking more than two existing ideas.

No matter how many existing ideas (individuals) will be piggybacked to generate new ideas (individuals), the cluster centers will have high probability to be chosen to generate new ideas (individuals) compared with the other non-cluster-center ideas (individuals) which usually can be chosen with small probability.

The coefficient $\xi(t)$ weights the contribution of randomly generated value to the new individual. Generally, large $\xi(t)$ value facilitates exploration while small $\xi(t)$ values facilitates exploitation. When global search capability is preferred, for example, at the beginning of search process, $\xi(t)$ should give large value, while when local search capability is preferred, for example, at the end of search process, $\xi(t)$ should give small value. One possible function for $\xi(t)$ is

$$\xi(t) = logsig\left(\frac{\frac{T}{2} - t}{k}\right) * random(t) \quad (3)$$

where *logsig*() is a logarithmic sigmoid transfer function, T is the maximum number of iterations, and t is the current iteration number, k is for changing *logsig*() function's slope, and *random*() is a random value within $(0,1)$.

Disruption

After two rounds of idea generation, the mindset of the brainstorming group usually will be narrowed and therefore it becomes more difficult, if not impossible, for them to come out different ideas efficiently. To further explore whether there are potential good ideas out there somewhere, in the brainstorming process, an object will be randomly picked up, and the brainstorming group will be asked to generate new ideas which are more or less related to the functions and appearance of the object. The purpose of this is to help the brainstorming group to disrupt from their current mindset, which is usually difficult to achieve. This disruption operation can be simulated by replacing selected ideas (individuals) with randomly generated individuals. Therefore, wider areas could be explored with high probability by utilizing disruption operation.

As shown in Figure 1, there are three rounds of idea generation. The first two rounds are identical while the third round serves as the purpose of disruption with the step Selecting an Object as Clue. To further modulate the operations, this disruption operation could be distributed and shared among all three rounds of idea generation. Figure 2 shows the modified flow chart of the brainstorming process, which includes three identical rounds of idea generation. Each round of idea generation is shown in Figure 3 in which the step Selecting an Object as Clue is changed to be the step

Figure 2. Flow chart of a brainstorming process

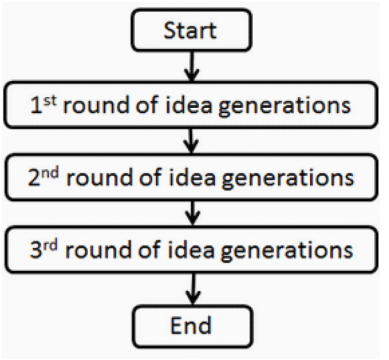
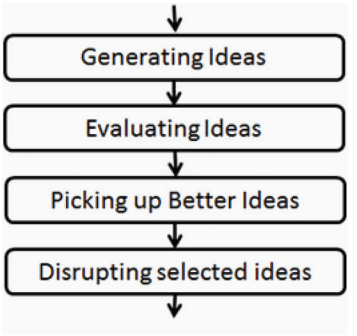


Figure 3. Flow chart of one round of idea generation



Disrupting Selected Ideas and it is put at the end of each round.

Selection

In a population-based optimization algorithm, generally speaking, if it is not because of specific requirements, the population size p is fixed and not changed during the algorithm running time. During each iteration, number of new individuals will be generated, say n $(n \geq p)$, therefore there will exist $p+n$ number of individuals, among which only p will be copied into the next iteration due to the fixed population size. Similar to other population-based algorithms, how to select p from $p+n$ individuals is critical to the optimization algorithm inspired by the brainstorming process. One simple way is that for each existing individual in the population, a new individual is generated. This pair of individuals is compared. The better one will be kept as the individual into the next iteration. Another way could be to randomly pick up p pairs of individuals from the $n+p$ individuals, and the better one of each pair will be kept into the next iteration.

To further take advantage of information embedded in each pair of individuals, crossover operations could also be applied to each pair of individuals to generate two new offspring. The best of the four will then be copied into the next iteration.

In each round of idea generation shown in Figure 3, one more step Selecting Ideas is inserted right below the step Generating Ideas. Figure 4 shows the new flow chart of each round of idea generation.

Figure 4. Flow chart of one round of idea generation

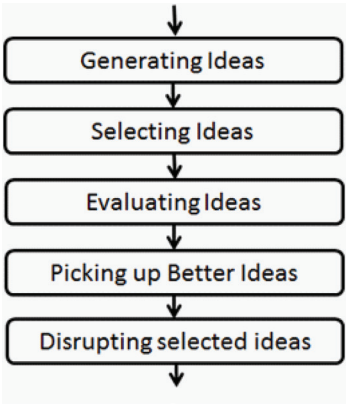
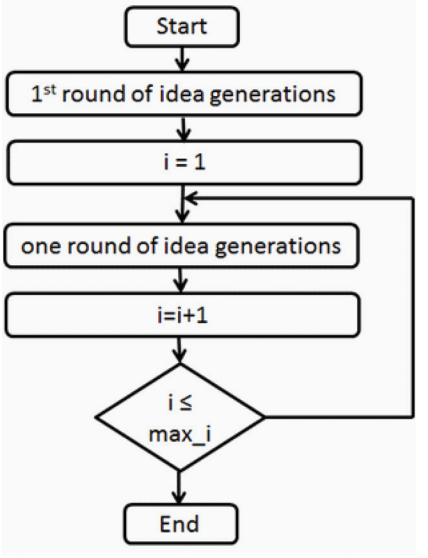
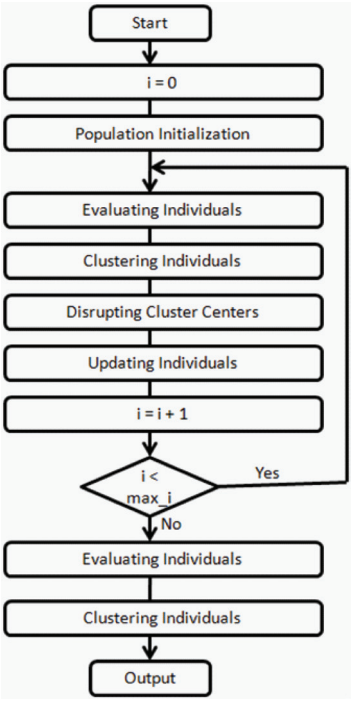


Figure 5. Flow chart of a brainstorming process



In practice, limited time will be taken for a brainstorming process, otherwise, the brainstorming group will be tired to generate new meaningful ideas efficiently. Usually as a good practice, a brainstorming process takes approximately 60 minutes. As shown in the Figure 1, there are only three rounds of idea generation in a brainstorming process. But for a model to be executed by computers, the number of rounds of idea generation can be as large as that we want. Figure 5 shows the flow chart for a brainstorming process that can be simulated by computers. In Figure 5, the step 1st Round of Idea Generations is the same as the step One Round of Idea Generation shown in Figure 5. The purpose to have the extra step 1st Round of Idea Generations at the beginning is to be similar to the Initialization step in population-based algorithms. The *max_i* is the maximum number of rounds of idea generation we want to conduct. Therefore totally, *max_i* rounds of idea generation will be conducted in

Figure 6. An implementation of BSO algorithm



the brainstorming process shown in the Figure 5.

By implementing Figure 5, a model or algorithm to mimic the human being brainstorming process can be built.

BRAIN STORM OPTIMIZATION ALGORITHM

According to the Figure 5, a brain storm optimization (BSO) algorithm can be designed by directly mapping the steps in the Figure 5. By some straightforward rearrangement, one possible flow chart of the BSO algorithm is shown in Figure 6. In Figure 6, there are five main operations among which three operations are unique to the BSO algorithm and the other two operations are similar to those in other evolutionary algorithms.

In the procedure of the Brain Storm Optimization (BSO) algorithm shown in the Figure 6, the first two steps are the initialization step and evaluation step which are the same as that in other swarm intelligence algorithms. In the initialization step, the population of individuals is usually uniformly and randomly initialized within the dynamic range of solution space. The population size n simulates the number of ideas generated in each round of idea generation in the brainstorming process. For the simplicity of the algorithm, the population size usually is set to be a constant number for all iterations in the BSO algorithm. In the evaluation step, each individual will be evaluated. An evaluation value (fitness) will be obtained to measure how good the individual as a potential solution to the problem to be solved. The third step is to cluster the population of individuals into several clusters. Different kind of clustering algorithms could be employed. In this paper, the k-means clustering algorithm will be used as the clustering algorithm. The disruption step randomly selects a cluster center and replace it with a randomly generated individual. This step

Figure 7. One implementation of updating individual operation

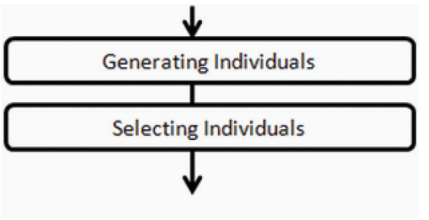
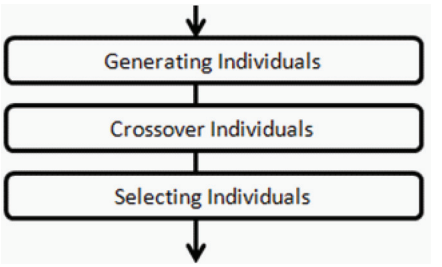


Figure 8. Another implementation of updating individual operation



will not be executed in every iteration, but will only be selected to execute with small probability.

The Updating Individuals step generally includes two sub-operations, i.e., Generating Individuals and Selecting Individuals, which is shown in Figure 7. As discussed in previous section, crossover operation could be utilized to further take advantage of existing search information. Figure 8 shows another possibility of the Updating Individuals operation which adds one additional sub-operation, i.e., crossover operation. One implementation of the BSO algorithm was introduced in Shi (2011) and is given in Table 3 here for convenience, in which the Updating Individuals operation shown in Figure 7 is implemented. By replacing the Step 6.d in Table 3 with two sub-steps shown in Table 4, another implementation of BSO algorithm can be achieved. To distinguish the two different implementations, the first one is noted as BSO-I and the second is noted as BSO-II for the purpose of description convenience. Intuitively, the BSO algorithms should be superior to other swarm intelligence algorithms, which are inspired by collective behaviors of inferior animals, because of the highest intelligence unique to human beings.

In the BSO algorithm, the number of cluster centers is usually set to be a small number, say $m=5$, and the number of generated individuals in each iteration is usually set to be a relatively large number, say $n=100$.

EXPERIMENTS AND DISCUSSIONS

Test Problems

To validate the brain storm optimization algorithms, ten benchmark functions listed in Table 5 are tested. Among them, the first five functions are unimodal functions and the remaining five functions are multimodal functions. They all are minimization problems with minimum zero. The third column in the Table 5 is the dynamic ranges for the ten benchmark functions, which have been used to test population-based algorithms in the literature. For each benchmark function, the tested BSO algorithm will be run 50 times to obtain reasonable statistical results.

Table 3. The procedure of the brain storm optimization algorithm in Shi (2011)

1. Randomly generate n potential solutions (individuals);
2. Evaluate the n individuals;
3. Cluster n individuals into m clusters by k-means clustering algorithm;
4. Rank individuals in each cluster and record the best individual as cluster center in each cluster;
5. Randomly generate a value between 0 and 1;
   a) If the value is smaller than a pre-determined probability $p_{5a,}$
      i. Randomly select a cluster center;
      ii. Randomly generate an individual to replace the selected cluster center;
   b) Otherwise, do nothing.
6. Generate new individuals
   a) Randomly generate a value between 0 and 1;
   b) If the value is less than a probability p_{6b},
      i. Randomly select a cluster with a probability p_{6bi};
      ii. Generate a random value between 0 and 1;
      iii. If the value is smaller than a pre-determined probability p_{6biii},
         1) Select the cluster center and add random values to it to generate new individual.
      iv. Otherwise randomly select an individual from this cluster and add random value to the individual to generate new individual.
   c) Otherwise randomly select two clusters to generate new individual
      i. Generate a random value;
      ii. If it is less than a pre-determined probability p_{6c}, the two cluster centers are combined and then added with random values to generate new individual;
      iii. Otherwise, two individuals from each selected cluster are randomly selected to be combined and added with random values to generate new individual.
   d) The newly generated individual is compared with the existing individual with the same individual index, the better one is kept and recorded as the new individual;
7. If n new individuals have been generated, go to step 8; otherwise go to step 6;
8. Terminate if pre-determined maximum number of iterations has been reached; otherwise go to step 2.

Table 4. Two sub-steps to replace step 6.d in table 3

d) The newly generated individual crossovers with the existing individual with the same individual index to generate two more individuals (offspring);
e) The four individuals are compared, the best one is kept and recorded as the new individual;

Simulations on *k*

In Shi (2011), the BSO-I algorithm was tested on two benchmark functions, i.e., the Sphere function and the Rastrigin function. The parameters are setup as that listed in Table 6. The purpose there is to validate the usefulness and effectiveness of the proposed BSO-I algorithm. Generally speaking, the parameter k determines the slope of the *logsig*() functions, therefore it determines the decreasing speed of the step-size $\xi(t)$ over iterations. Different k should have different impacts on the performance of BSO algorithms. In order to test the impact of k on BSO performance, we change the k value while all other parameter values are kept to be the same as that listed in Table 6. For this purpose, again only one unimodal function, Sphere function, and one multimodal function, Rastrigin function, are utilized. The dimension of the two functions is set to be 20.

Table 7 gives the simulation results. The results given in the Table 7 are mean, best, worst function values and their variance at the final iteration over *50* runs. From the Table 7, it can be observed that generally there is no single

Table 5. Benchmark functions tested in this paper

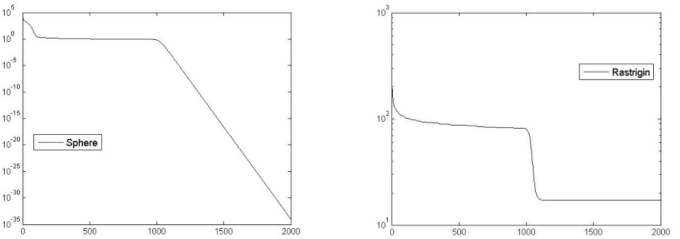
Function	Expressions	Range
Sphere	$f_1 = \sum_{i=1}^{d} x_i^2$	$[-100,100]^d$
Schwefel's P221	$f_2 = \max_i\{\lvert x_i\rvert\}$	$[-100,100]^d$
Step	$f_3 = \sum_{i=1}^{d} (\lfloor x_i + 0.5\rfloor)^2$	$[-100,100]^d$
Schwefel's P222	$f_4 = \sum_{i=1}^{d} \lvert x_i\rvert + \prod_{i=1}^{d} \lvert x_i\rvert$	$[-10,10]^d$
Quartic Noise	$f_5 = \sum_{i=1}^{d} i x_i^4 + \text{random}[0,1)$	$[-1.28,1.28]^d$
Ackely	$f_6 = -20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right) + 20 + e$	$[-32,32]^d$
Rastrigin	$f_7 = \sum_{i=1}^{d} [x_i^2 - 10\cos(2\pi x_i) + 10]$	$[-5.12,5.12]^d$
Rosenbrock	$f_8 = \sum_{i=1}^{d-1} [100(x_{i+1} - x_i^{\,2})^2 + (x_i - 1)^2]$	$[-30,30]^d$
Schwefel's P226	$f_9 = -\sum_{i=1}^{d} (x_i \sin(\sqrt{\lvert x_i\rvert}) + 418.9829d$	$[-500,500]^d$
Griewank	$f_{10} = \frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d} \cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-600,600]^d$

parameter k value with which the BSO algorithm can have the best performance. From the Table 7, relatively speaking, unimodal function (Sphere function) prefers relatively small k value while multimodal function (Rastrigin function) prefers relatively large k value. By considering the robustness of the BSO algorithm and from the results given in Table 7 itself, generally speaking, a good choice for the parameter k is *25* as a tradeoff between unimodal function and multimodal function. The obtained mean function values over *2000* iterations with parameter k *=25* are shown in Figure 9. From the Figure 9, it can be observed that the BSO-I with k *=25* can converge fast when solving the Sphere function and Rstrigin function. In all simulations, we will set the parameter k to be *25* with all other parameters are set as the same as that in Table 6.

Simulations on BSO-I Algorithm

The BSO-I algorithm is tested on the ten benchmark functions listed in Table 5 to illustrate

Figure 9. Obtained mean minimum values vs. iterations for BSO-I with k = 25



a) 20-D Sphere Function *b) 20-D Rastrigin Function*

Table 6. Set of parameters for BSO algorithm

n	m	p_{5a}	p_{6b}	p_{6biii}	p_{6c}	k	Max_ iteration	μ	σ
100	5	0.2	0.8	0.4	0.5	20	2000	0	1

Table 7. Simulation results of BSO-I with different k

function	k	mean	best	worst	variance
Sphere	10	1.20381E-11	2.55674E-86	5.27132E-10	5.61316E-21
	20	2.30827E-43	1.24079E-43	3.17853E-43	2.5931E-87
	25	9.4726E-35	5.55931E-35	1.33105E-34	3.78414E-70
	30	5.35092E-29	3.01328E-29	7.6509E-29	1.30974E-58
	40	7.70746E-22	4.18609E-22	1.10552E-21	2.74116E-44
	50	1.60782E-17	7.60191E-18	2.22015E-17	1.18045E-35
Rastrigin	10	17.11636	5.969754	29.84873	28.93288
	20	18.00875	8.954632	31.83866	20.98068
	25	17.17298	6.964713	23.879	13.24541
	30	17.15308	7.959667	29.84871	26.04389
	40	15.81984	6.964713	24.87396	17.95025
	50	16.21782	8.954632	25.8689	16.85928

the effectiveness and efficiency of the BSO-I algorithm instead of only two benchmark functions in Shi (2011). Each function is tested with three different dimension setting, *10*, *20*, and *30*, respectively. The experimental results are given in Tables 8 and 9 for unimodal functions and multimodal functions, respectively. From the Table 8, it can be seen that good results can be achieved by BSO-I algorithm, and the results also show that the BSO-I is robust and reliable when it is applied to solve benchmark unimodal functions. From the Table 9, it can be seen that good results can be obtained for function f_6, relatively good results can be ob-

Table 8. Simulation results of BSO-I on unimodal functions

Function	Dimension	mean	best	worst	variance
f_1	10	1.3989E-35	3.90855E-36	2.71203E-35	2.75801E-71
	20	9.77845E-35	6.11475E-35	1.37856E-34	3.55418E-70
	30	2.66069E-34	1.79135E-34	3.59892E-34	2.02141E-69
f_2	10	2.31285E-18	1.49658E-18	3.11619E-18	1.47169E-37
	20	5.05671E-18	3.69394E-18	6.40744E-18	4.41064E-37
	30	0.000235	3.18538E-08	0.001718355	1.55583E-07
f_3	10	0	0	0	0
	20	0	0	0	0
	30	0	0	0	0
f_4	10	9.28917E-18	5.51341E-18	1.21942E-17	1.81665E-36
	20	3.4224E-17	2.63097E-17	4.25525E-17	1.03733E-35
	30	1.9978E-06	5.84794E-17	9.94736E-05	1.97869E-10
f_5	10	0.000424	4.52215E-05	0.001140455	6.14016E-08
	20	0.002636	0.000613	0.008465	2.8024E-06
	30	0.00835095	0.001967	0.020706	1.33183E-05

Table 9. Simulation results of BSO-I on multimodal functions

Function	Dimension	mean	best	worst	variance
f_6	10	4.44089E-15	4.44089E-15	4.44089E-15	0
	20	4.44089E-15	4.44089E-15	4.44089E-15	0
	30	5.93303E-15	4.44089E-15	7.99361E-15	3.13741E-30
f_7	10	3.502256	0	5.969754	1.949178
	20	17.75005	8.954632	26.86387	15.12629
	30	34.56484	13.92943	51.7378	51.65143
f_8	10	6.330642	2.587793	29.36235	11.77892
	20	21.60337	15.83735	87.11474	255.4539
	30	42.02786	25.91331	296.7523	2073.832
f_9	10	1350.782	454.0165	2270.172	192322.2
	20	3012.657	1598.991	4501.054	570878.2
	30	4951.779	3652.088	6771.33	563448.4
f_{10}	10	1.35123	0.497182	2.21245	0.158512
	20	0.058446	0	0.9467	0.022289
	30	0.010777	0	0.056496	0.000163

tained for functions f_7 and f_8, but not relatively good results are obtained for f_9 which is in general a difficult function to optimize, and for function f_{10}, good results can be obtained for it with dimension *20* and *30*, but not with dimension*10*, for which only relatively good results are obtained instead.

Simulation on BSO-II Algorithm

The BSO-II algorithm further exploits the search areas by generating two new offspring through utilizing crossover operation to crossover the newly generated individual with the existing individual with the same individual index. The BSO-II is applied to the ten benchmark functions with dimensions *10*, *20*, and *30*, respectively. The simulation results are given in Tables 10 and 11 for unimodal functions and multimodal functions, respectively. From the Table 10, we can observe that good results can be achieved by the BSO-II algorithm, and the results also show that the BSO-II is robust and reliable when it is applied to solve benchmark unimodal functions. From the Table 11, it can be seen that good results can be obtained for function f_6, relatively good results can be obtained for functions f_7 with dimension 30 and f_8, but not relatively good results are obtained for f_9 which is in general a difficult function to optimize, and for function f_{10}, reasonable good results can be obtained. Compared with the observation from the BSO-I algorithm, very good results (the optimum) can be obtained for f_7 with dimension *10* and *20*. For f_7 with dimension *30*, the best results over *50* runs is *0* which is the optimum of the problem, but the worst and variance over *50* runs are *3.979836* and *1.010551*, which indicates that the BSO-II is better than the BSO-I, but it is still not robust when solving f_7 function.

To further compare the BSO-I and BSO-II algorithm, Figures 10 through 19 show curves which display the average evaluation function values over *50* runs vs. iterations for the ten benchmark functions tested. From the figures, it can be easily seen that the BSO-II algorithm performs better than the BSO-I algorithm for all the benchmark functions with all three different dimensions except the Griewank function with dimension *20* and *30*. For Griewnak function with dimension *10*, the BSO-I can't obtain very good results but the BSO-II could. Therefore, even for the Griewank function, the BSO-II could be a better choice compared with the BSO-I algorithm. For function f_9, even though still not very good results are obtained by BSO-II, but BSO-II performs much better than BSO-I does.

Diversity

During each iteration, the population of individuals is clustered into m clusters. Individuals in each cluster are scattered with different distribution over iterations. To measure and monitor the distribution of individuals in each cluster, the following average intra-cluster distance is defined

$$d_c(x_i, x_j) = \left\| x_i - x_j \right\| \tag{4a}$$

$$\tilde{d}_c(x_i, x_j) = \frac{d_c(x_i, x_j)}{\left\| a - b \right\|} \tag{4b}$$

$$D_c = \frac{2}{q(q-1)} \sum_{i=1}^{q} \sum_{j=i+1}^{q} \tilde{d}(x_i, x_j) \tag{4c}$$

where q is the number of individuals in a cluster; $d(x_i,x_j)$ is the Euclidean distance between individual x_i and x_j; a and b are dynamic range; $\tilde{d}(x_i,x_j)$ is the normalized Euclidean distance between individual x_i and x_j; D_c is the normalized distance for a cluster. For $m=5$, there will be *5* intra-cluster diversities. In addition to m average intra-cluster distances, there will be one average inter-cluster distance to measure and/or monitor the distribution of cluster centers. The formula for calculating average intra-cluster distance can also be utilized to calculate the average inter-cluster distance except that here the x_i is the ith cluster center and number of individuals is m.

Over iterations, the number of individuals in each cluster will change. To measure and

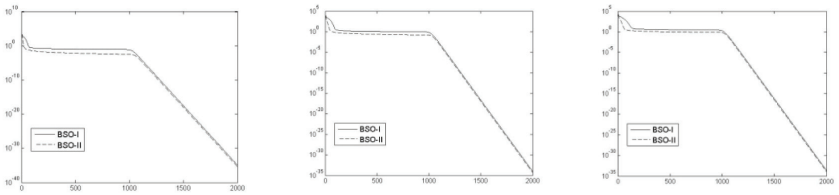
Table 10. Simulation results of BSO-II on unimodal functions

Function	Dimension	mean	best	worst	variance
f_1	10	4.56E-36	2.61244E-36	7.53912E-36	1.13477E-72
	20	4.54E-35	2.98742E-35	6.19235E-35	7.00667E-71
	30	1.33E-34	9.34047E-35	1.65808E-34	2.53466E-70
f_2	10	1.52E-18	1.10811E-18	1.94476E-18	3.80565E-38
	20	3.86E-18	3.23175E-18	4.44916E-18	8.94695E-38
	30	5.85E-18	4.80866E-18	6.91767E-18	2.62081E-37
f_3	10	0	0	0	0
	20	0	0	0	0
	30	0	0	0	0
f_4	10	4.76E-18	3.30555E-18	6.17314E-18	5.02845E-37
	20	2.13E-17	1.54076E-17	2.52198E-17	5.11026E-36
	30	4.49E-17	3.56463E-17	5.22311E-17	1.57E-35
f_5	10	8.85E-05	3.18035E-05	0.000256579	1.69387E-09
	20	0.000319	0.000104	0.000853	2.24337E-08
	30	0.000766	0.000176	0.001733	1.00554E-07

Table 11. Simulation results of BSO-II on multimodal functions

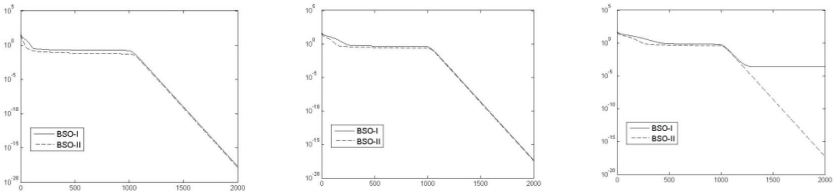
Function	Dimension	mean	best	worst	variance
f_6	10	4.16E-15	8.88178E-16	4.44089E-15	9.47921E-31
	20	4.44E-15	4.44089E-15	4.44089E-15	0
	30	4.44E-15	4.44089E-15	4.44089E-15	0
f_7	10	0	0	0	0
	20	0	0	0	0
	30	0.855665	0	3.979836	1.010551
f_8	10	4.558798	2.019811	9.069095	0.862945
	20	28.514436	15.49093	83.89686	604.2519
	30	34.06948	25.85653	128.9086	505.6776
f_9	10	56.23811	0.000127	236.8768	5855.616
	20	499.023	118.4386	927.8028	48176.18
	30	1128.729	335.5784	1993.748	157231
f_{10}	10	0.150697	0.022151	0.531254	0.019883
	20	0.311937	0.017241	1.519837	0.110482
	30	0.090445	0	0.568672	0.011172

Figure 10. Mean function evaluation values vs. iterations of sphere function



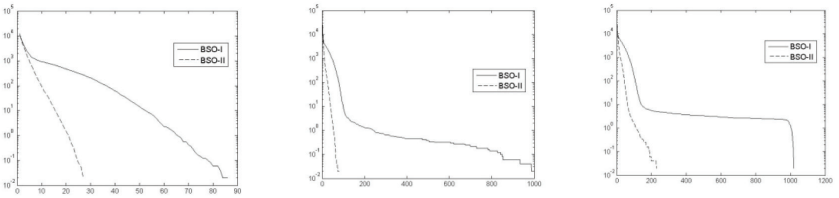
a) 10-D *b) 20-D* *c) 30-D*

Figure 11. Mean function evaluation values vs. iterations of Schwefel's P221



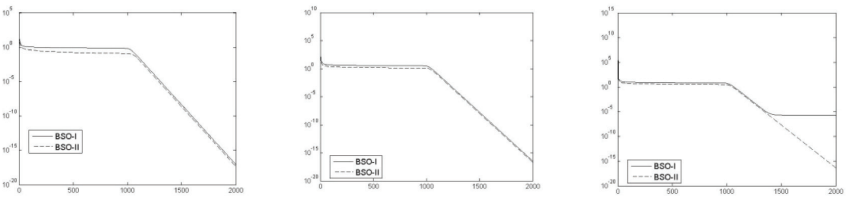
a) 10-D *b) 20-D* *c) 30-D*

Figure 12. Mean function evaluation values vs. iterations of step function



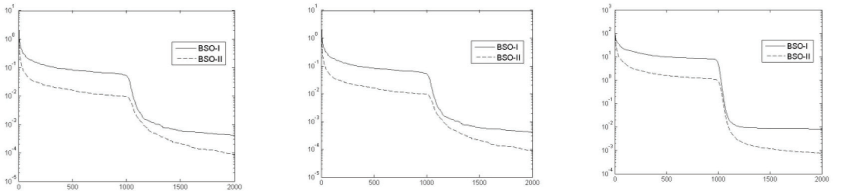
a) 10-D *b) 20-D* *c) 30-D*

Figure 13. Mean function evaluation values vs. iterations of Schwefel's P222



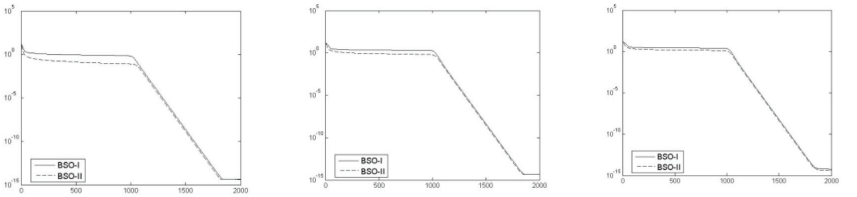
a) 10-D *b) 20-D* *c) 30-D*

Figure 14. Mean function evaluation values vs. iterations of quartic noise



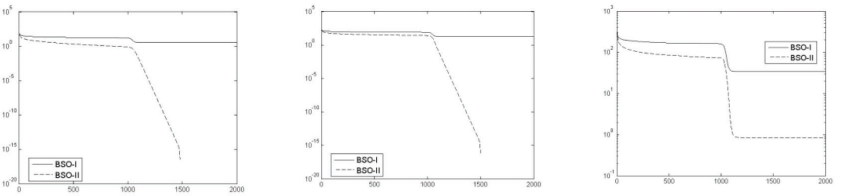
a) 10-D *b) 20-D* *c) 30-D*

Figure 15. Mean function evaluation values vs. iterations of Ackely function



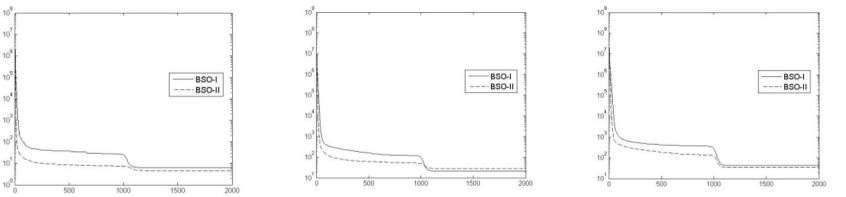
a) 10-D *b) 20-D* *c) 30-D*

Figure 16. Mean function evaluation values vs. iterations of Rastrigin function



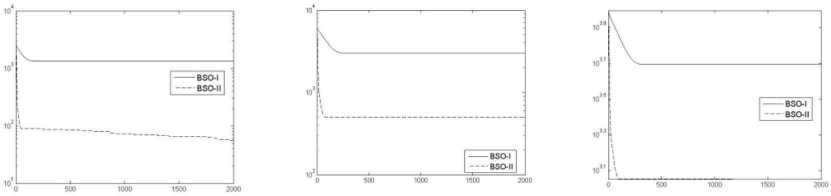
a) 10-D *b) 20-D* *c) 30-D*

Figure 17. Mean function evaluation values vs. iterations of Rosenbrock function



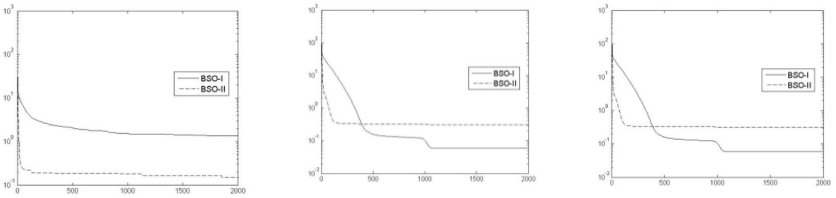
a) 10-D *b) 20-D* *c) 30-D*

Figure 18. Mean function evaluation values vs. iterations of Schwefel's P226



a) 10-D *b) 20-D* *c) 30-D*

Figure 19. Mean function evaluation values vs. iterations of Griewank function



a) 10-D *b) 20-D* *c) 30-D*

monitor the distribution of number of individuals in each cluster over whole population, the following inter-cluster diversity is defined

$$D_v = \sum_{i=1}^{m} \frac{(n_i - \bar{n})^2}{m} \quad ,\bar{n} = \frac{\sum_{i=1}^{m} n_i}{m} \tag{5}$$

where m is the number of clusters, n_i is the number of individuals in the ith cluster. D_v is similar to the definition of variance for distribution of number of individuals in each cluster among the population.

Another similar definition of the inter-cluster diversity can be defined as

$$D_e = -\sum_{i=1}^{m} p_i \log(p_i), \quad p_i = \frac{n_i}{n} \tag{6}$$

where m is the number of clusters, n_i is the number individuals in the ith cluster. Therefore p_i is the percentage of individuals that the ith cluster has over the population. D_e is similar to the definition of information entropy. Therefore, it can be looked as a measurement of information entropy for the population. When all the individuals are located in one cluster, the D_e has the smallest value, which is *0*; when all the individuals are equally distributed into each cluster, D_e has the largest value, which is *log(m)*. If *m =5, D_e = log(5) = 0.699*.

Tables 12 and 13 give the results of average inter-cluster distance D_c and inter-cluster diversity D_e for ten tested benchmark functions at the end of BSO-I running. Tables 14 and 15 give the results of average inter-cluster distance D_c and inter-cluster diversity D_e for ten tested benchmark functions at the end of BSO-II running. Figures 20 through 29 show mean average inter-cluster distance over 50 runs vs. iterations for the ten benchmark functions, respectively. From both the Tables 12 through 15 and Figures 20 through 29, it could be easily observed that the average inter-cluster distance quickly decreases over iterations and gets to very

Table 12. Simulation results of D_c and D_e for BSO-I on unimodal functions

F	d	D_c				D_e			
		mean	short	long	variance	mean	small	large	variance
f_1	10	3.99E-20	3.02E-20	5.45E-20	2.87E-41	0.674	0.576	0.697	0.00056
	20	7.73E-20	5.84E-20	9.83-20	7.02E-41	0.658	0.566	0.697	0.000893
	30	9.92E-20	7.47E-20	1.16E-19	1.10E-40	0.645	0.473	0.695	0.001978
f_2	10	4.6E-20	3.31E-20	5.58E-20	2.71E-41	0.671	0.583	0.695	0.000528
	20	9.11E-20	6.98E-20	1.15E-19	1.17E-40	0.653	0.579	0.693	0.000653
	30	2.2E-19	1.40E-19	3.57E-19	2.36E-39	0.624	0.488	0.694	0.002807
f_3	10	0.006908	0.005538	0.008269	2.80E-07	0.685	0.666	0.697	7.604E-05
	20	0.007145	0.005921	0.007997	1.89E-07	0.666	0.572	0.695	0.000567
	30	0.006818	0.005485	0.007595	2.15E-07	0.653	0.499	0.693	0.001402
f_4	10	4.44E-19	3.24E-19	5.95E-19	3.44E-39	0.674	0.608	0.699	0.000481
	20	8.54E-19	6.73E-19	1.12E-18	1.01E-38	0.652	0.567	0.698	0.000925
	30	1.12E-18	8.86E-19	1.55E-18	1.83E-38	0.640	0.554	0.694	0.001363
f_5	10	0.06774	0.031242	0.107543	0.000272	0.609	0.484	0.681	0.002202
	20	0.041977	0.016624	0.071656	0.000163	0.604	0.394	0.686	0.003644
	30	0.029224	0.010858	0.04638	6.94E-05	0.591	0.389	0.682	0.004305

Table 13. Simulation results of D_c and D_e for BSO-I on multimodal functions

F	d	D_c				D_e			
		mean	short	long	variance	mean	small	large	variance
f_6	10	1.01E-16	7.83E-17	1.24E-16	6.71E-35	0.677	0.617	0.698	0.00023
	20	1.01E-16	7.28E-17	1.25E-16	1.30E-34	0.642	0.560	0.695	0.001086
	30	1.14E-16	1.01E-18	2.21E-16	5.69E-33	0.613	0.384	0.692	0.003186
f_7	10	2.57E-10	1.31E-16	4.39E-10	1.16E-10	0.607	0.450	0.692	0.002966
	20	1.90E-10	4.56E-17	7.13E-10	3.82E-20	0.604	0.448	0.690	0.002892
	30	1.22E-10	1.22E-17	7.10E-10	3.49E-20	0.589	0.378	0.695	0.004818
f_8	10	1.15E-17	6.43E-19	5.77E-17	1.06E-34	0.622	0.505	0.688	0.002294
	20	2.22E-17	1.17E-18	8.01E-17	3.61E-34	0.600	0.456	0.686	0.003408
	30	2.75E-17	1.49E-18	1.81E-16	1.05E-33	0.588	0.349	0.692	0.00625
f_9	10	1.91E-09	5.68E-17	3.89E-09	7.92E-19	0.618	0.115	0.685	0.007802
	20	2.32E-09	3.22E-16	4.14E-09	8.62E-19	0.603	0.378	0.692	0.004574
	30	2.16E-09	6.82E-17	4.30E-09	1.30E-18	0.593	0.097	0.683	0.00893
f_{10}	10	1.21E-11	7.22E-19	8.07E-11	3.72E-22	0.589	0.097	0.693	0.01162
	20	5.01E-11	5.05E-18	9.57E-11	6.74E-22	0.602	0.362	0.689	0.003937
	30	5.21E-11	3.854E-16	9.28E-11	5.74E-22	0.610	0.506	0.696	0.002307

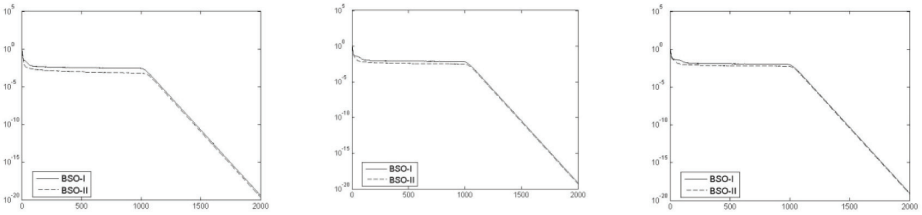
Table 14. Simulation results of D_c and D_e for BSO-II on unimodal functions

F	d	D_c				D_e			
		mean	short	long	variance	mean	small	large	variance
f_1	10	2.35E-20	1.80E-20	3.00E-20	5.86E-42	0.674	0.609	0.696	0.000435
	20	5.3E-20	4.39E-20	6.41E-20	2.37E-41	0.676	0.568	0.696	0.000443
	30	7.5E-20	6.48E-20	8.77E-20	3.51E-41	0.674	0.612	0.695	0.000363
f_2	10	3.16E-20	2.50E-20	4.16E-20	1.36E-41	0.675	0.597	0.697	0.000592
	20	7.4E-20	5.56E-20	8.36E-20	3.47E-41	0.658	0.586	0.694	0.000782
	30	1.06E-19	9.13E-20	1.22E-19	5.64E-41	0.658	0.540	0.696	0.001079
f_3	10	0.007209	0.005138	0.008104	2.81E-07	0.685	0.652	0.698	8.749E-05
	20	0.007577	0.006727	0.008262	1.37E-07	0.685	0.610	0.697	0.000214
	30	0.00797	0.007136	0.008711	1.77E-07	0.677	0.622	0.695	0.000298
f_4	10	2.51E-19	1.77E-19	3.58E-19	1.42E-39	0.667	0.546	0.697	0.000904
	20	5.58E-19	4.59E-19	8.08E-19	5.04E-39	0.658	0.509	0.699	0.001228
	30	8.14E-19	5.89E-19	1.11E-18	9.52E-39	0.662	0.567	0.696	0.001039
f_5	10	0.073914	0.052208	0.104325	0.000129	0.649	0.509	0.695	0.00143
	20	0.066032	0.044808	0.095853	0.000149	0.638	0.515	0.692	0.001513
	30	0.062562	0.041369	0.102545	0.000142	0.624	0.326	0.688	0.003219

Table 15. Simulation results of D_c and D_e for BSO-II on multimodal functions

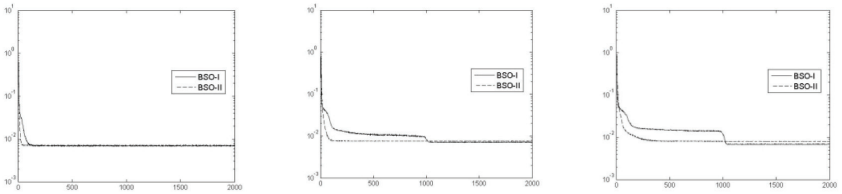
F	d	D_c				D_e			
		mean	short	long	variance	mean	small	large	variance
f_6	10	9.83E-17	1.13E-18	1.25E-16	8.10E-34	0.684	0.616	0.698	0.00025
	20	1.17E-16	7.84E-17	1.47E-16	1.68E-34	0.655	0.570	0.693	0.000735
	30	1.02E-16	5.82E-17	1.38E-16	3.12E-34	0.597	0.419	0.691	0.00392
f_7	10	4.63E-10	3.95E-10	5.28E-10	1.12E-21	0.688	0.665	0.698	5.111E-05
	20	4.9E-10	4.10E-10	5.46E-10	8.55E-22	0.681	0.615	0.698	0.000203
	30	4.85E-10	4.03E-10	5.68E-10	8.28E-22	0.663	0.590	0.698	0.000578
f_8	10	1.93E-13	3.12E-19	9.66E-12	1.87E-24	0.589	0.411	0.687	0.004572
	20	1.52E-16	7.17E-19	6.67E-16	1.98E-32	0.599	0.434	0.683	0.003304
	30	1.38E-16	8.33E-19	6.48E-16	2.28E-32	0.585	0.468	0.668	0.002776
f_9	10	2.13E-09	0	3.22E-09	6.53E-19	0.571	0.097	0.685	0.024857
	20	3.42E-09	0	4.95E-09	1.38E-18	0.595	0.097	0.693	0.024992
	30	5.79E-09	3.21E-09	7.72E-09	9.25E-19	0.634	0.443	0.697	0.003643
f_{10}	10	3.24E-11	1.51E-19	6.07E-11	5.18E-22	0.623	0.421	0.694	0.002855
	20	5.89E-11	1.35E-18	1.04E-10	1.04E-21	0.625	0.493	0.694	0.002619
	30	8.63E-11	2.99E-20	1.24E-10	1.40E-21	0.631	0.499	0.691	0.002384

Figure 20. Mean average inter-cluster vs. iterations of sphere function



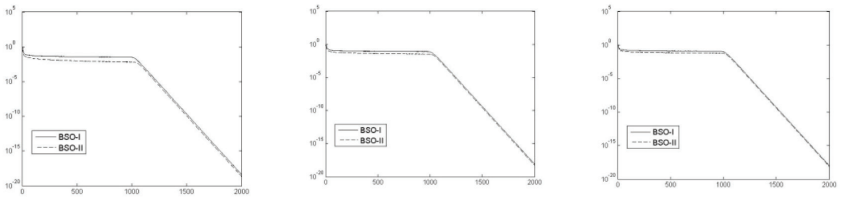
a) 10-D *b) 20-D* *c) 30-D*

Figure 21. Mean average inter-cluster distance vs. iterations of Schwefel's P221



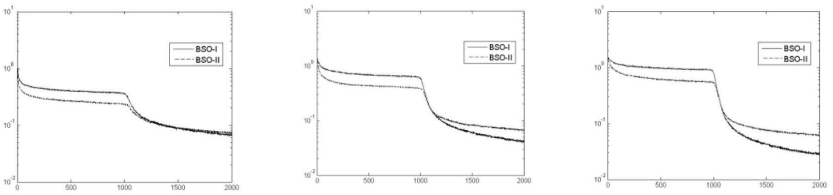
a) 10-D *b) 20-D* *c) 30-D*

Figure 22. Mean average inter-cluster distance vs. iterations of step function



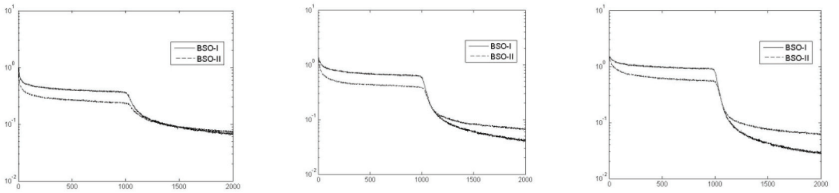
a) 10-D *b) 20-D* *c) 30-D*

Figure 23. Mean average inter-cluster distance vs. iterations of Schwefel's P222



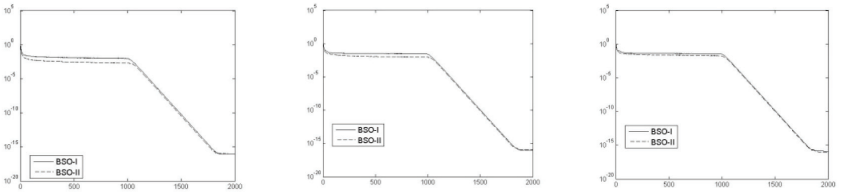
a) 10-D *b) 20-D* *c) 30-D*

Figure 24. Mean average inter-cluster distance vs. iterations of quartic noise



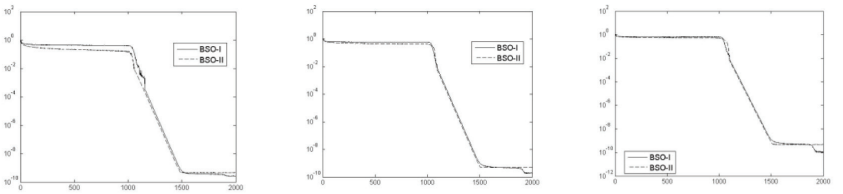
a) 10-D *b) 20-D* *c) 30-D*

Figure 25. Mean average inter-cluster distance vs. iterations of Ackely function



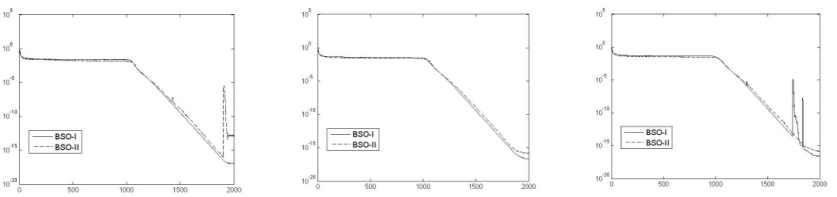
a) 10-D *b) 20-D* *c) 30-D*

Figure 26. Mean average inter-cluster distance vs. iterations of Rastrigin function



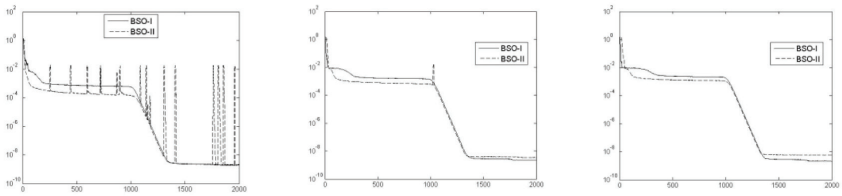
a) 10-D *b) 20-D* *c) 30-D*

Figure 27. Mean average inter-cluster distance vs. iterations of Rosenbrock function



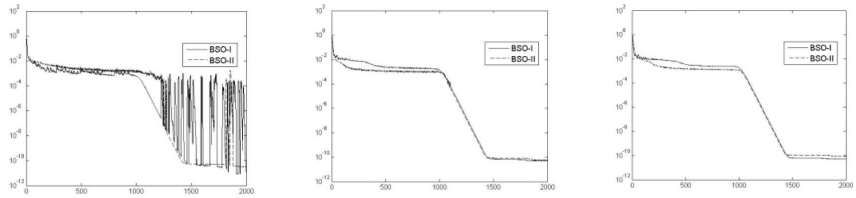
a) 10-D *b) 20-D* *c) 30-D*

Figure 28. Mean average inter-cluster distance vs. iterations of Schwefel's P226



a) 10-D *b) 20-D* *c) 30-D*

Figure 29. Mean average inter-cluster distance vs. iterations of Griewank function

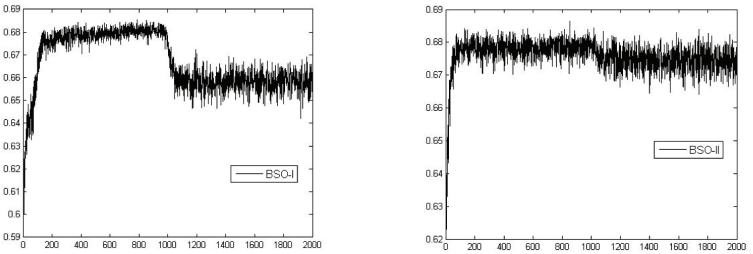


a) 10-D *b) 20-D* *c) 30-D*

small values way before reaching the prefixed maximum iteration number for both BSO algorithms, which indicates that *m* clusters move close to each other very quickly, and therefore the algorithms may lose their search capabilities quickly, may converge quickly, or may be stuck in (local) optima quickly. By double checking the cluster centers over iterations, the same observation can be obtained. That tells us that when the situation occurs, further improvement could be achieved by randomly move away from current cluster centers and at the same time increase the step-size to a relatively large value which then will be dynamically adjusted according to the formula (3).

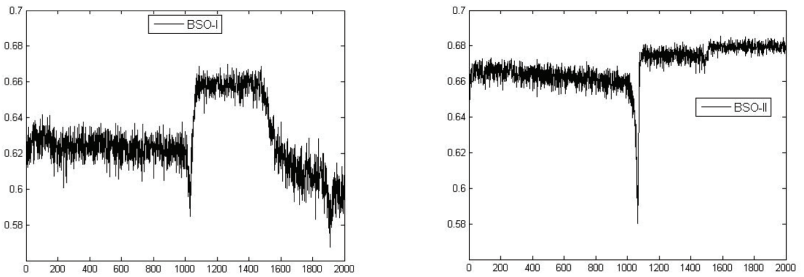
The mean inter-cluster diversities of *50* runs over iterations for all benchmark functions seem to have similar behaviors except function f_9 with dimension *10* and *20*. Figures 30 and 31 display the curves of mean inter-cluster diversities over *50* runs vs. iterations for function f_1 as an example for unimodal functions and for function f_7 as an example for multimodal functions. Figure 32 displays the curves of mean inter-cluster diversities over *50* runs vs. iterations for function f_9 with dimension *20*. From Figures 30 and 31, the mean inter-cluster diversities tend to have relatively large values, which indicate that the population of individuals is generally well-uniformly divided into *m* clusters. This may be because the fixed number of clusters and the k-means clustering algorithm with randomly selecting *k* individuals as initial cluster centroid positions are used over iterations in the implementation of the BSO algorithms. If different initialization method for k-means clustering algorithm or a different clustering algorithm especially those with unfixed number of clusters such as the self-organizing feature map is utilized, the mean inter-cluster diversity may behave quite different. From Figure 32 and Table 15, it can be seen that toward the end of BSO-II running for function f_9, the number of individuals in each cluster is not uniformly distributed anymore, but

Figure 30. Mean D_e over 50 runs vs. iterations for sphere function with dimension d = 20



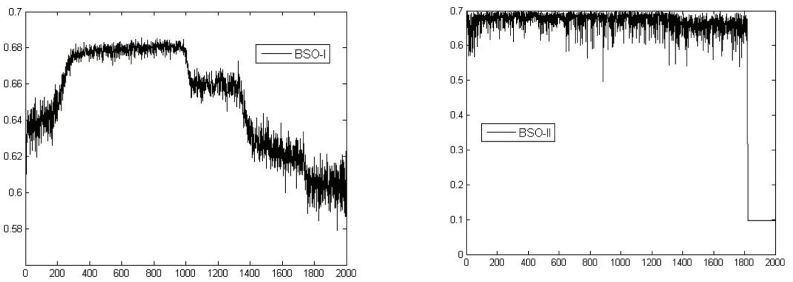
a) BSO-I *b) BSO-II*

Figure 31. Mean D_e over 50 runs vs. iterations for Rastrigin function with dimension d = 20



a) BSO-I *b) BSO-II*

Figure 32. Mean D_e over 50 runs vs. iterations for Schwefel's p226 with dimension d = 20



a) BSO-I *b) BSO-II*

clustered into one cluster with other *4* clusters with only *1* individual, in which the

$$D_e = -\left(4 * \frac{1}{100} * log_{10}\left(\frac{1}{100}\right) + \frac{96}{100} * log_{10}\left(\frac{96}{100}\right)\right) = 0.097$$

This may indicate that when the above situation occurs the portion of clustering all individuals into m clusters in the BSO algorithm may have very little, if not none, contribution to the algorithm's search capability.

CONCLUSION

In this paper, we first modeled the human brainstorming process, then introduced two versions of Brain Storm Optimization algorithms It is natural to believe that BSO algorithms should be superior to the optimization algorithms inspired by collective behavior of injects such as ants, birds, *etc*. because the BSO algorithms were inspired by the human brainstorming process. The proposed BSO algorithms were implemented and tested on ten benchmark functions, of which five are unimodal functions and the other five are multimodal functions. Simulation results showed that both BSO algorithms performed reasonably well, and BSO-II performs better than BSO-I does in general. Furthermore, average inter-cluster distance D_c and inter-cluster diversity D_e were defined to measure and monitor the distribution of cluster centroids and the information entropy of the BSO population. Simulation results on D_c showed that further performance improvement for the BSO algorithms could be achieved by taking advantage of information revealed by the D_c and or D_e, which is one of future research directions.

Good optimization algorithms for solving complicated and nonlinear optimization problems should have the capability to converge in order to find better and better solutions, but at the same time, it should have the capability to diverge in order to escape from local optima which are not good enough solutions for the problem to be solved. The BSO algorithm during each iteration involves two opposite operations. One is to converge or contract by utilizing clustering methods to converge to the *m* cluster centers. Another is to diverge or expand by adding noise to generate new individuals. Depending on the amplitude of noise, different scales of areas can be searched by the BSO algorithm. Therefore, the BSO algorithms naturally include contraction and expansion operations during each iteration by design. It should be a good choice for solving complicated and nonlinear optimization problems.

ACKNOWLEDGMENT

This paper is partially supported by National Natural Science Foundation of China under Grant Number 60975080, and by the Suzhou Science and Technology Project under Grant Number SYJG0919.

REFERENCES

Bastos-Filho, C. J. A., De Lima Neto, F. B., Lins, A. J. C. C., Nascimento, A. I. S., & Lima, M. P. (2008). A novel search algorithm based on fish school behavior. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics* (pp. 2646-2651).

de Castro, J. N., & Von Zuben, F. J. (1999). *Artificial immune systems: Part I -Basic theory and applications* (Tech. Rep. No. DCA-RT 01/99). Brazil, Campinas: School of Computing and Electrical Engineering, State University of Campinas.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man. Cybernetics B*, *26*(2), 29–41. doi:10.1109/3477.484436

Eberhart, R. C., & Shi, Y. (2007). *Computational intelligence, concepts to implementation* (1st ed.). San Francisco, CA: Morgan Kaufmann.

Eusuff, M., & Lansey, K. (2006). Shuffled frog-leaping algorithm: A memetic meta-heuristic for discrete optimization. *Engineering Optimization*, *38*(2), 129–154. doi:10.1080/03052150500384759

Fogel, L. J. (1962). Autonomous automata. *Industrial Research*, *4*, 14–19.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Kohonen, T., & Honkela, T. (2007). Kohonen network. *Scholarpedia*, *2*(1), 1568. doi:10.4249/scholarpedia.1568

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability* (pp. 281-297).

Nock, R., & Nielsen, F. (2006). On weighting clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(8), 1–13. doi:10.1109/TPAMI.2006.168

Osborn, A. F. (1963). *Applied imagination: Principles and procedures of creative problem solving* (3rd ed.). New York, NY: Charles Scribner's Son.

Passino, K. M. (2010). Bacterial foraging optimization. *International Journal of Swarm Intelligence Research*, *1*(1), 1–16. doi:10.4018/jsir.2010010101

Pavlyukevich, I. (2007). Lévy flights, non-local search and simulated annealing. *Journal of Computational Physics*, *226*, 1830–1844. doi:10.1016/j.jcp.2007.06.008

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart, Germany: Frommann-Holzboog.

Shah-Hosseini, H. (2009). The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm. *International Journal of Bio-inspired Computation*, *1*(1-2), 71–79. doi:10.1504/IJBIC.2009.022775

Shi, Y. (2011, June 11-15). Brain storm optimization algorithm. In Y. Tan, Y. Shi, Y. Chai, & G. Wang (Eds.), *Proceedings of the Second International Conference on Advances in Swarm Intelligence*, Chongqing, China (LNCS 6728, pp. 303-309).

Shi, Y., & Eberhart, R. C. (1998, May 4-9). A modified particle swarm optimizer. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, Anchorage, AK.

Shi, Y., & Eberhart, R. C. (2008). Population diversity of particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, Hong Kong, China.

Shi, Y., & Eberhart, R. C. (2009). Monitoring of particle swarm optimization. *Frontiers of Computer Science in China*, *3*(1), 31–37. doi:10.1007/s11704-009-0008-4

Smith, R. (2002). *The 7 levels of change* (2nd ed.). Arlington, VA: Tapeslry Press.

Theodoridis, S., & Koutroumbas, K. (2006). *Pattern recognition* (3rd ed.). New York, NY: Academic Press.

Tovey, C. (2004). The honey bee algorithm: A biological inspired approach to internet server optimization. *Engineering Enterprise, the Alumni Magazine for ISyE at Georgia Institute of Technology*, 13-15.

Xu, R., & Wunsch, D. II. (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, *16*(3), 645–678. doi:10.1109/TNN.2005.845141

Yang, X. (2008). *Nature-inspired metaheuristic algorithms*. Beckington, UK: Luniver Press.

Yao, X., Liu, Y., & Lin, G. (1997). Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, *3*, 82–102.

Yuhui Shi received the PhD degree in electronic engineering from Southeast University, Nanjing, China, in 1992. He is currently a Professor with the Department of Electrical and Electronic Engineering, Xi'an Jiaotong-Liverpool University, Suzhou, China. His current research interests include computational intelligence techniques (including swarm intelligence) and their applications. Dr. Shi is the Editor-in-Chief of the International Journal of Swarm Intelligence Research *and an Associate Editor of the* IEEE Transactions on Evolutionary Computation. *He is the Chair of the IEEE Task Force on Swarm Intelligence.*