Artif Intell Rev (2011) 36:267–284
DOI 10.1007/s10462-011-9213-2

A discrete shuffled frog optimization algorithm

M. T. Vakil Baghmisheh · Katayoun Madani · Alireza Navarbaf

Published online: 9 April 2011

Abstract The shuffled frog leaping (SFL) optimization algorithm has been successful in solving a wide range of real-valued optimization problems. In this paper we present a discrete version of this algorithm and compare its performance with a SFL algorithm, a binary genetic algorithm (BGA), and a discrete particle swarm optimization (DPSO) algorithm on seven low dimensional and five high dimensional benchmark problems. The obtained results demonstrate that our proposed algorithm, i.e. the DSFL, outperforms the BGA and the DPSO in terms of both success rate and speed. On low dimensional functions and for large values of tolerance the DSFL is slower than the SFL, but their success rates are equal. Part of this slowness could be attributed to the extra bits used for data coding. By increasing number of variables and the required precision of answer, the DSFL performs very well in terms of both speed and success rate. For high dimensional problems, for intrinsically discrete problems, also when the required precision of answer is high, the DSFL is the most efficient method.

Keywords Optimization · Binary genetic algorithm · Discrete shuffled frog algorithm · Discrete particle swarm algorithm

1 Introduction

The difficulties of using classic mathematical optimization methods on large-scale engineering problems have contributed to the development of alternative methods. The linear programming and the dynamic programming techniques, for example, often fail to solve NP-hard problems with a large number of variables and non-linear objective functions

M. T. Vakil Baghmisheh (✉) · K. Madani · A. Navarbaf
ICT Research Center, Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran
e-mail: mvakil@tabrizu.ac.ir

K. Madani
e-mail: katayoun.madani@tabrizu.ac.ir

A. Navarbaf
e-mail: navarbaf88@ms.tabrizu.ac.ir

(Lovbjerg 2002). To overcome these shortcomings, researchers have developed evolutionary algorithms (EAs) to find suboptimal solutions. The evolutionary algorithms are stochastic search methods that mimic the metaphor of biological evolution and/or the social behavior of species.

The very first group of evolution-based techniques introduced in the literature, were the genetic algorithms (GAs) (Holland 1975). The GAs were developed based on the Darwinian principle of "the survival of the fittest" and the natural process of evolution through exchange of genes and mutation. Based on its demonstrated ability to reach near-optimal solutions for large scale and non-linear problems, the GAs techniques have been used in many applications in science and engineering (Al-Tabtabai and Alex 1999; Grierson and Khajehpour 2002; Hegazy 1999). Despite their benefits, the GAs may require long processing time for a near optimal solution to evolve. Also, not all the problems lend themselves well to a solution with the GAs (Joglekar and Tungare 2003).

In an attempt to reduce the processing time and to improve the quality of the solutions, particularly to avoid being trapped in local optima, other EAs have been introduced during the past 20 years. An example of an evolutionary algorithm, which has been demonstrated to be successful on high dimensional benchmark problems, is the particle swarm optimization (PSO). The PSO is a population-based optimization method first introduced by Kennedy and Eberhart (1995). It was inspired by the swarming behavior as is displayed by a flock of birds, a school of fish, or even human social behavior being influenced by other individuals. The PSO consists of a swarm of particles moving in n-dimensional real-valued search space of possible problem solutions. Every particle has a position vector encoding a candidate solution to the problem and a velocity vector. Moreover, each particle contains a small memory that stores its own best position visited so far and a neighborhood/global best position obtained through communication with its neighbor particles. The information about good solutions spreads through the swarm, and thus the particles tend to move towards promising areas in the search space. In every iteration, the velocity is updated and the particle moves to a new position. This new position is calculated as the sum of the previous position and the velocity vector. In 1997, Eberhart and Kennedy proposed a discrete version of the PSO algorithm (DPSO). In a binary space, a particle may be seen to move to different corners of a hypercube by flipping various numbers of bits (Eberhart and Kennedy 1997); in a discrete space, velocity defines the probability of the position component to take the value one. The DPSO algorithm can solve the simple discrete optimization problems effectively. But, for some complicated problems, it may fail to converge or need too long time to find a good solution (Xu et al. 2006).

Another example of the evolutionary algorithms is the shuffled frog leaping algorithm (SFL). The SFL algorithm (Elbeltagi et al. 2005; Eusuff and Lansey 2003), in essence, combines the benefits of the genetic-based memetic algorithms (MAs) and the social behavior-based PSO algorithms. In the SFL, the population consists of a set of frogs (solutions) that is partitioned into subsets referred to as memeplexes. The different memeplexes are considered as different cultures of frogs, each performing a local search.

Within each memeplex, the individual frogs hold ideas, that can be influenced by the ideas of other frogs, and evolve through a process of memetic evolution. After a predefined number of memetic evolution steps, ideas are passed among memeplexes in a shuffling process. The local search and the shuffling processes continue until satisfying the stopping criteria.

In this paper, we present a discrete shuffled frog algorithm. Essentially, it combines the benefits of the BGA and the DPSO algorithm for improving the search process, but the main idea is like the SFL algorithm. Then its performance is compared against the SFL, the DPSO and the BGA on 12 benchmark functions.

The rest of paper is organized as follows. In Sect. 2, we give a short review of the utilized BGA. Considering that the DSFL algorithm is obtained by combining the methods of the DPSO and the SFL algorithms, the DPSO algorithm is reviewed in Sect. 3. Then in Sect. 4 the SFL algorithm is presented and the proposed DSFL algorithm is introduced in Sect. 5. In Sect. 6, the experimental results on seven low dimensional and five high dimensional benchmark functions are presented and analyzed. In Sect. 7, the general conclusion is presented. The descriptions of benchmark functions are given in Appendix A.

2 The binary genetic algorithm

The BGA is inspired by biological systems' improved fitness through the process of evolution (Holland 1975). A solution to a given problem is represented in the form of a string, called chromosome, consisting of a set of elements, called genes, which hold a set of values for the optimization variables (Goldberg 1989). The BGA utilizes some operators whose performances are similar to those of natural biological process, such as selection, crossover and mutation.

The BGA starts with a random population of solutions (chromosomes). The fitness of each chromosome is determined by evaluating the objective function. To simulate the natural survival of the fittest, a predetermined percent of chromosomes that have lower fitness are eliminated, and the remaining better ones receive a chance to find a mate and to produce offspring. The selection operator determines the parent individuals, which should mate. There are many ways for choosing parents (Haupt and Haupt 2004). The most common methods are roulette wheel and tournament selection methods. Since in our simulations we have used roulette wheel selection method, it is described here, briefly.

In the roulette wheel selection, first a fitness value is assigned to every remaining individual, as a function of its either cost value or its rank in the poll. That is to say, better individuals with lower cost values, receive higher fitness values. Then a selection probability is attributed to every individual. In our simulations, we have used the following simple formula:

$$P_n = \frac{N_{keep} - n + 1}{\sum_{n=1}^{N_{keep}} n} \tag{1}$$

where N_{keep} is number of remaining chromosomes, and n is the rank of chromosome in the poll. Then the cumulative probabilities ($\sum_{i=1}^{n} P_i$) are listed in an increasing order and a random number in the interval [0, 1] is generated. Starting from top of the list, the first chromosome with a cumulative probability that is greater than the random number is selected as the first parent; the second parent is selected similarly.

After selection of parents, the crossover operator should be applied to produce two offspring by exchanging some genetic information between the parents. There are some variants of crossover operator such as single-point, two-point, uniform, arithmetic etc. In our simulations, we have used a two-point crossover, which is described below.

In the two-point crossover, two random crossover points are selected throughout the chromosomes. Then the parents swap the bits staying between these two points. Alternatively, one of the three parts of the chromosomes could be randomly selected for swapping.

This process of selection and reproduction is repeated until the number of offspring becomes equal to the number of eliminated chromosomes; by adding the offspring to the population, its size becomes equal to the initial size.

The mutation is the third operator, which is applied on the population (parents and their offspring) excluding the best chromosome; this policy of excluding the best chromosome from mutation is called elitism, without which the BGA normally fails to converge. A simple mutation operator selects a few percent of the genes of the population randomly and flips their values from zero to one and vice versa (Elbeltagi et al. 2005). In our simulations, we set the mutation rate at 0.15. The benefit of mutation is introducing new genetic material to the population, thus it prevents the algorithm from early convergence and stagnation around local minima (Goldberg 1989). The obtained population is called a new generation whose members are evaluated and the stopping criteria are checked.

More details on the mechanism of the BGA could be found in (Al-Tabtabai and Alex 1999; Goldberg 1989). Four main parameters which affect the performance of the BGA are population size, number of generations or function evaluations, crossover type and mutation rate. A larger population size (i.e. hundreds of chromosomes) and large number of generations (thousands) increase the likelihood of obtaining a global optimum solution, but substantially increase processing time (Elbeltagi et al. 2005).

3 The discrete particle swarm optimization algorithm

Chronologically the continuous variant of the PSO was introduced first, and the discrete variants were introduced based on the concepts used in the continuous variant. Hence, we first present the continuous variant briefly, then the discrete variant is described.

The continuous PSO algorithm is a population-based, self-adaptive search technique introduced by Kennedy and Eberhart (1995). The development of this method was based on the simulation of simplified social behaviors such as fish schooling and bird flocking or even human social behaviour being influenced by other individuals. Similar to other population-based optimization methods (e.g. genetic algorithms), the PSO starts with a random population of individuals (particles) in the search space. It finds the global best solution by simply adjusting the trajectory of each individual towards its own best position and towards the best position of the neighborhood or the entire swarm. The PSO method has become very popular due to its simplicity, efficiency, and fast convergence properties. There are two major variants of the PSO algorithm, local and global. The global variant of the original PSO algorithm is as follows (Kennedy and Eberhart 1995):

1. Initialize a population of M particles with random positions and velocities in the D-dimensional search space.
2. Evaluate the fitness function for all particles.
3. For all particles, compare particle's current position with its personal best position achieved so far $PB_i = (pb_{i1}, pb_{i2}, \ldots, pb_{iD})$. If its current position is better than PB_i, set PB_i equal to the current position.
4. Identify the best-ever position visited by any particle in the swarm as the global best position GB.
5. Update velocity and position vectors of all particles using Eqs. (2) and (3).
6. If stopping criteria are not met, go to step 2 and repeat the algorithm.

The velocity and position vectors are updated based on these formulas:

$$V_{id}^{n+1} = V_{id}^{n} + c_1 \cdot rand() \cdot (gb_{id}^{n} - x_{id}^{n}) + c_2 \cdot Rand() \cdot (pb_{id}^{n} - x_{id}^{n}); \quad \text{for } d = 1, \ldots, D \tag{2}$$

$$X_i^{n+1} = X_i^{n} + V_i^{n+1} \tag{3}$$

where V_i represents the velocity of particle i, n is iteration number, c_1 and c_2 are the acceleration constants which are positive numbers in the range [0.5, 2], $rand()$ and $Rand()$ are two random numbers in the range [0, 1]; X_i represents the ith particle's position. PB and GB represent the best-ever position achieved by the particle itself and all particles in the swarm, respectively.

In the local version of the PSO, the global best position (GB) is replaced with the neighborhood best position of each particle (NB). It has been suggested that the global version of PSO converges faster than the local version (Shi 2004).

In 1997, Eberhart and Kennedy proposed the first discrete version of the PSO algorithm (DPSO) (Eberhart and Kennedy 1997). Since 1997 many variants of DPSO algorithm have been presented. Afshinmanesh et al. (2005) introduced a method based on the theory of immunity in biology. Al-kazemi and Mohan (2002b) proposed the Discrete Multi-Phase Particle Swarm Optimization (DiMuPSO) algorithm. The main feature of the DiMuPSO is utilizing multiple groups of particles with different goals that are allowed to change with time, i.e. alternately moving towards or away from the best solutions found recently. The DiMuPSO also enforces steady improvement of solutions qualities by accepting only those moves which improve fitness (Al-kazemi and Mohan 2002a,b; Al-kazemi 2002). Xu et al. (2006) proposed three sub-swarm discrete particle swarm optimization algorithm (THSDPSO) and introduced two ways of handling the position of a particle. Tseng and Liao (2007) developed the DPSO for solving lot-streaming flow shop scheduling problem. The new DPSO improves the existing DPSO by introducing an inheritance scheme, inspired by a genetic algorithm into particles construction. Xu et al. (2008) presented an improved DPSO based on cooperative swarms, which partitions the search space into lower dimensional subspaces. The k-means split scheme and regular split scheme are applied to split the solution vector into swarms. Then the swarms optimize the different components of the solution vector cooperatively. Sharaf and El-Gammal (2009) developed a novel discrete optimization approach to optimally solve the optimization problem of power system shunt filter design based on Discrete Multi Objective Particle Swarm Optimization (MOPSO) technique to ensure harmonic current reduction and noise mitigation on electrical utility grid.

The DPSO algorithm which is used as a benchmark for comparing the performance of new algorithms is given below (Eberhart and Shi 2001).

1. Initialize a population of M particles with random positions and velocities in the D-dimensional search space.
   Remark- The components of velocity vector are interpreted as the probability measures for the position bits to take the value one (See step 5).
2. Evaluate the fitness function for all particles.
3. For all particles, compare particle's current position with its personal best position achieved so far $PB_i = (pb_{i1}, pb_{i2}, \ldots, pb_{iD})$. If the current position is better than PB_i, then set PB_i equal to the current position.
4. Identify the best-ever position achieved by any particle in the swarm as the global best position GB.
5. The velocity of all particles and their new positions are updated according to the following three equations:
   for $d = 1, \ldots, D$ and $i = 1, \ldots, M$:

$$v_{id}^{n+1} = \xi \cdot (\omega v_{id}^{n} + c_1 \cdot r_1 \cdot (pb_{id}^{n} - x_{id}^{n}) + c_2 \cdot r_2 \cdot (gb_{d}^{n} - x_{id}^{n})) \tag{4}$$

$$S(v_{id}^{n+1}) = 1/1 + \exp(-v_{id}^{n+1}) \tag{5}$$

$$x_{id}^{n+1} = \begin{cases} 1 & \text{if } rand() \leq S\left(v_{id}^{n+1}\right) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where ξ is the constriction factor which is used to limit the maximum velocity, c_1 and c_2 are two positive constants called acceleration coefficients ($c_1 = c_2 = 2$), which respectively determine the impacts of particle's personal experience and social experience on the future movements of the particle; r_1 and $r_2$are two random numbers uniformly distributed between 0 and 1, $rand()$ is also a random number between 0 and 1 which is different for every component, ω is called the inertia weight (Kennedy and Spears 1998) and is calculated from Eq. (7):

$$\omega = 0 \cdot 7 + \frac{rand()}{2} \cdot \ (1 - \exp(-iter)) \tag{7}$$

6. If stopping criteria are not met, go to step 2 and repeat the algorithm.

Equation (4) updates the velocity of a particle using three parts: The first part is the current velocity of the particle; the second is the distance between the particle's current position and the best personal position visited by the particle (personal experience); the last part is the distance between the particle's current position and the global best position visited by any particle so far, which reflects sharing the knowledge and co-operation among particles.

Equations (5) and (6) indicate that every position component of the particle is determined using the corresponding velocity component as a probability measure.

Implementation of this algorithm requires enforcing boundary limits on v values. If v in one dimension exceeds the boundary limit, it is set to $v_{\max}$, this parameter controls the convergence rate and can prevent the algorithm from divergence (Clerc and Kennedy 2002; Zhang et al. 2003). As such, the main parameters used in the DPSO technique are: the population size, maximum number of generation cycles or function evaluations, number of bits for each variable, ξ (the constriction factor), c_1, c_2, ω (the inertia weight) and the maximum velocity.

The suggested values for constriction factor and maximum velocity are 0.729 and 7 respectively (Eberhart and Shi 2001). Both c_1 and c_2 are equal to 2 and ω is calculated using Eq. (7).

4 The shuffled frog leaping algorithm

The SFL algorithm (Elbeltagi et al. 2005; Eusuff and Lansey 2003), in essence, combines the benefits of the genetic-based memetic algorithms and the PSO algorithms.

Let us consider a group of frogs in a swamp. The swamp has a number of stones at discrete locations where the frogs can leap onto. The goal of all frogs is to find the stone with the maximum amount of available food as quickly as possible through improving their memes. The frogs can communicate with each other, and can improve their memes based on the received information. Improvement of the memes results in changing an individual frog's position by adjusting its leaping direction and step size. A shuffling strategy allows for the exchange of information between local groups to move toward a global optimum.

The difference and potential advantage of the memetic representation over genetic algorithms is that information is passed among all individuals in the population, whereas in a BGA only parent–sibling interactions are allowed.

The SFL algorithm is as follows:

1. Initialize a population of P frogs with random positions in the search space.
2. The fitness function for each frog is evaluated.

3. The frogs are sorted in a descending order according to their fitness.
4. The whole population is divided into m memeplexes, each containing l frogs (i.e. $P = m \times l$). In this process, the first frog goes to the first memeplex, the second frog goes to the second memeplex, frog m goes to the mth memeplex, and frog $m + 1$ goes back to the first memeplex, etc.
5. Within the ith memeplex, the frogs with the best and the worst fitness are identified as XB_i and XW_i, respectively. Also, the best-ever position visited by any frog is identified as the global best position GB.
6. The frog with the worst fitness in every memeplex (not all the frogs) is improved using steps A to C.
   A. A process similar to the PSO algorithm is applied to improve $XW_i = (xw_{i1}, \ldots, xw_{iD})$ of each memeplex. Accordingly, the position of the frog is adjusted as follows:

$$\text{change in frog position: } (\Delta X_i) = rand() \cdot (XB_i - XW_i) \tag{8}$$

$$xw_{id}^{n+1} = xw_{id}^{n} + \Delta x_{id}; \quad -\Delta x_{\max} \le \Delta x_{id} \le \Delta x_{\max} \tag{9}$$

   where $rand()$ is a random number between 0 and 1, n is iteration number, and $\Delta x_{\max}$ is the maximum allowed change in a frog-position's component. If this process produces a better solution, it replaces the worst frog. Otherwise, go to the next step.
   B. The calculations in Eqs. (8) and (9) are repeated but with respect to the global best frog (i.e. GB replaces XB_i). If no improvement becomes possible, go to the next step.
   C. A new random solution is generated to replace the worst frog XW_i.

   Go to step 5 and repeat the improvement process for a predefined number of times (s).
7. The memeplexes are shuffled.
8. The frogs are sorted in a descending order according to their fitness values.
9. If stopping criteria are not met, go to step 4 and repeat the algorithm.

Accordingly, the main parameters of the SFL are: number of frogs p, number of memeplexes m, number of processing cycles for each memeplex before shuffling s, maximum number of shuffling iterations or function evaluations, and maximum step size for every component $\Delta x_{\max}$.

Different settings were experimented to determine proper values for parameters. Six memeplexes, 10 improvement cycles per memeplex before shuffling, and 0.0125 as the maximum step size were found suitable to obtain good solutions.

5 The discrete shuffled frog leaping algorithm

In the DSFL algorithm each frog is represented by a number of bits and the only difference with the SFL is in the cycle of improving the worst frog position within each memeplex. The improving cycle has four steps, in the first step it uses a method which in concept is somehow similar to the DPSO algorithm, and for the second and third steps it uses the operators of the BGA, i.e. mutation and crossover.

To improve the position of the worst frog (XW) of every memeplex, follow these four steps:

1. Use Eq. (10) to calculate the speed vector of the worst frog VW_i:
for$d = 1, \ldots, N_{bit}$:

$$vw_{id}^{n+1} = \xi(\omega \cdot vw_{id}^{n} + c_1 \cdot r_1 \cdot (pb_{id}^{n} - xw_{id}^{n}) + k \cdot \mu_1 \cdot c_2 \cdot r_2 \cdot (gb_{d}^{n} - xw_{id}^{n}) + \mu_2 \cdot c_3 \cdot r_3 \cdot (xb_{id} - xw_{id}^{n}) \quad (10)$$

where i denotes the worst frog of ith memeplex, n represents the iteration number, PB_i is the best position visited previously by the worst frog of ith memeplex and XB_i is the position of the best frog in ith memeplex, and ξ is the constriction factor; c_1, c_2 and c_3 are three positive constants called acceleration coefficients ($c_1 = c_2 = c_3 = 2$); r_1, r_2 and r_3 are three random numbers uniformly distributed between 0 and 1. μ_1 and μ_2 are called the influence factors, μ_1 reflects the influence of the global best position on the worst frog and μ_2 reflects the influence of the best position of any memeplex imposed on the worst frog. As a rule μ_1 and μ_2 are positive decimal fractions. The default values of μ_1 and μ_2 are as $\mu_1 = \mu_2 = 0.5$. k reflects the movement direction, which is selected randomly, thus if $k = 1$ the frog moves towards the global best position, else $k = -1$ and it moves in the opposite direction. ω is called the inertia weight (Kennedy and Spears 1998), and is calculated from Eq. (7).
The position of the frog is determined using Eq. (11):

$$xw_{id}^{n+1} = hardlim(xw_{id}^{n} + vw_{id}^{n+1}) \quad (11)$$

where

$$hardlim(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

If this process produces a better solution, it replaces the worst frog; otherwise go to the next step.
2. A mutation operator is applied on the position of the worst frog. In the case of improvement, the resulted position is accepted; otherwise go to the next step.
3. A crossover operator is applied between the worst frog of the memeplex and the globally best position. The worst frog is replaced if its fitness is improved; otherwise go to the next step.
4. The worst frog is replaced randomly.

After a predefined number of improvement cycles, memeplexes are shuffled, and if stopping criteria are not met, the algorithm is repeated.

Accordingly, the main parameters of DSFL are: number of frogs P, number of memeplexes m, number of processing cycles on each memeplex before shuffling, number of shuffling iterations (or function evaluations), number of bits for any variable, mutation rate, crossover type, the constriction factor, acceleration coefficients and influence factors.

Based on some primary experimental results, the suitable values were found as follows: number of frogs and number of bits for each variable are 60 and 10, respectively, number of processing cycles on each memeplex before shuffling is 10, number of memeplexes is 6. The values of other parameters have been mentioned before.

6 The simulation results

We applied our proposed algorithm on seven low dimensional and five high dimensional test functions. The explanations of these functions are given in Appendix A.

Then, we compared the performance of the proposed algorithm with those of three evolutionary algorithms, i.e. the SFL, the DPSO and the BGA.

The population size for all algorithms is 60, and number of bits for all variables in the discrete algorithms is 10.

The algorithms were executed on each problem for 30 runs. The performances of these four algorithms were compared using two criteria: (1) success rate; (2) average number of function evaluations in successful runs.

Remark - To measure the speed of algorithms we do not use number of generation cycles, because number of function evaluations in different cycles of different algorithms are different.

Stopping criteria: In all experiments, the algorithm is stopped if one of the following three criteria is satisfied:

1. The objective function reaches the target value of 0.001 (i.e. reaches within an acceptable distance from the known optimum value) or: $|f - f^*| \leq 0.001$ where f^* is the optimal cost value and f is best cost value obtained so far.
2. The number of function evaluations reaches to a maximum limit, which is 20,000 and 50,000 for low dimensional and high dimensional test functions, respectively.
3. No improvement is observed in the objective function for 30 consecutive iterations.

6.1 The simulation results on low dimensional test functions

As mentioned before, we selected seven low dimensional test functions ranging from 2 to 5 dimensions. (See Appendix A).

It can be seen from Table 1 that on LF_1, the DSFL, the SFL and the DPSO achieve a success rate of 100% with all answer precisions. Also the BGA achieves a success rate of 100% with larger tolerances, but only 80% with a tolerance of 10^{-3}. Although with large tolerances the DSFL is seven times slower than the SFL, but with small tolerances of answer the speed of the DSFL is comparable to that of the SFL. In comparison to the BGA, with lower precision the BGA is faster than the DSFL and their success rates are equal, but with higher precision, the DSFL is twice faster and achieves higher success rate. In comparison to the DPSO, the DSFL is always faster.

The above analysis suggests that where a high precision for the answer is needed the DSFL is a proper approach.

On LF_4 function, all four algorithms have achieved a success rate of 100% with all degrees of answer precision. The DSFL is the second fast algorithm, after the SFL. Although at lower precision the speed of algorithms do not differ that much, if an answer with a high precision is needed the convergence rate of the SFL and DSFL are three to six times faster than those of the BGA and the DPSO.

On LF_2 function, the success rate of the DSFL is 100% with all precisions of answer, although it is three times slower than the SFL. Also the success rate of the SFL is considerably lower than others. Putting the SFL aside, we notice that when a higher precision is required, i.e. 10^{-2} and 10^{-3}, the DSFL becomes the fastest algorithm.

On LF_3 function, with low precision of answer, the performances of all four algorithms are much the same. By increasing the required answer precision, the success rate of the DSFL remains untouched (100%), but those of the other three algorithms deteriorate.

On LF_5 function, the DSFL's performance is only lower than that of the SFL in terms of speed. Further the DPSO and the BGA fail to find the answer if the tolerance is set smaller than 10^{-2}.

Table 1 The success rate of four algorithms on low dimensional test functions with specified tolerance

Function	Algorithm	Tolerance			
		0.5	0.1	0.01	0.001
LF1	DSFL	100% (3,730)	100% (5,000)	100% (6,346)	100% (8,936)
	SFL	100% (580)	100% (1,098)	100% (2,084)	100% (6,944)
	DPSO	100% (4,280)	100% (6,780)	100% (11,400)	100% (14,640)
	BGA	100% (1,680)	100% (2,390)	100% (7,020)	80% (18,420)
LF2	DSFL	100% (6,603)	100% (10,150)	100% (12,707)	100% (15,296)
	SFL	90% (2,656)	80% (3,987)	50% (4,789)	40% (5,944)
	DPSO	100% (6,540)	100% (10,140)	100% (12,960)	70% (15,600)
	BGA	100% (1,880)	100% (1,740)	80% (18,300)	60% (19,140)
LF3	DSFL	100% (907)	100% (931)	100% (1,087)	100% (1,647)
	SFL	100% (980)	100% (997)	100% (1,005)	80% (1,710)
	DPSO	100% (879)	100% (994)	80% (1,021)	80% (2,640)
	BGA	100% (1,074)	90% (1,243)	60% (1,766)	50% (2,398)
LF4	DSFL	100% (166)	100% (643)	100% (3,704)	100% (4,500)
	SFL	100% (122)	100% (555)	100% (1,833)	100% (3,315)
	DPSO	100% (180)	100% (1,020)	100% (1,860)	100% (15,900)
	BGA	100% (220)	100% (1,520)	100% (3,540)	100% (18,300)
LF5	DSFL	100% (6,494)	100% (7,788)	100% (10,344)	100% (12,888)
	SFL	100% (540)	100% (720)	100% (917)	100% (1,364)
	DPSO	80% (13,260)	10% (18,784)	0% (0)	0% (0)
	BGA	100% (17,500)	70% (19,989)	0% (0)	0% (0)
LF6	DSFL	100% (10,830)	100% (10,970)	10% (11,000)	10% (15,888)
	SFL	100% (5,730)	30% (7,820)	0% (0)	0% (0)
	DPSO	60% (17,340)	20% (19,260)	0% (0)	0% (0)
	BGA	0% (0)	0% (0)	0% (0)	0% (0)
LF7	DSFL	100% (604)	100% (6,072)	30% (12,089)	10% (16,800)
	SFL	100% (3,301)	70% (5,981)	0% (0)	0% (0)
	DPSO	60% (7,340)	40% (9,470)	0% (0)	0% (0)
	BGA	50% (2,070)	50% (4,056)	0% (0)	0% (0)

The numbers in parentheses are the average function evaluations over successful runs

On LF_6 and when the tolerance is 0.5, only the SFL has a better performance (in terms of speed); the BGA cannot find the answer with any precision; the SFL and the DPSO fail to find the answer if the precision is 10^{-2} or smaller.

On LF_7, although by increasing the precision of answer the performance of the DSFL deteriorates in terms of both success rate and speed, deterioration of other algorithms' performances are much worse; in a way that with a tolerance of 10^{-2} their success rates are zero.

In brief, the DSFL is slower than the SFL but faster than the DPSO and the BGA. By increasing the precision of answer its success rate becomes much better than that of the SFL. Since, for intrinsically discrete problems, the SFL could not be used, the DSFL remains the superior algorithm for any required precision of answer.

Table 2 The success rate of four algorithms on high dimensional test functions ($D = 10$)

Function	Algorithm	Tolerance			
		0.5	0.1	0.01	0.001
HF1	DSFL	100% (5,509)	100% (11,695)	100% (13,402)	100% (15,520)
	SFL	100% (1,884)	100% (4,739)	100% (4,829)	100% (7,019)
	DPSO	100% (17,820)	100% (20,100)	100% (31,260)	100% (39,000)
	BGA	100% (26,790)	100% (33,500)	50% (39,270)	0% (0)
HF2	DSFL	100% (5,257)	100% (7,073)	60% (8,248)	20% (10,120)
	SFL	100% (1,190)	100% (1,989)	100% (2,028)	40% (3,870)
	DPSO	100% (6,420)	100% (9,300)	50% (9,980)	10% (16,780)
	BGA	100% (18,180)	60% (19,020)	30% (21,980)	0% (0)
HF3	DSFL	100% (12,942)	100% (15,145)	100% (19,189)	100% (23,730)
	SFL	100% (8,677)	100% (10,366)	100% (13,907)	100% (16,753)
	DPSO	100% (21,300)	100% (24,252)	100% (31,596)	100% (38,508)
	BGA	0% (0)	0% (0)	0% (0)	0% (0)
HF4	DSFL	100% (22,306)	100% (27,920)	100% (35,485)	100% (46,039)
	SFL	100% (18,420)	100% (27,330)	100% (39,281)	100% (46,218)
	DPSO	0% (0)	0% (0)	0% (0)	0% (0)
	BGA	0% (0)	0% (0)	0% (0)	0% (0)
HF5	DSFL	100% (11,222)	100% (15,155)	100% (17,680)	100% (19,637)
	SFL	100% (8,226)	100% (11,302)	100% (13,719)	100% (14,751)
	DPSO	100% (29,028)	100% (33,564)	100% (40,692)	100% (46,572)
	BGA	100% (33,612)	100% (40,908)	100% (46,965)	0% (0)

The numbers in parentheses are the average function evaluations over successful runs

In the next subsection, we investigate the authenticity of this conclusion on high dimensional problems.

6.2 The simulation results on high dimensional test functions

The performance of our proposed algorithm is evaluated on five high dimensional benchmark problems. The dimensions of these problems range from 10 to 100 (see Appendix A). The obtained results are given in Tables 2, 3, 4 and 5.

Table 2 illustrates the obtained results for number of variables equal to 10. In this table, on HF_1, the BGA is the slowest algorithm among these four algorithms. The performance of the BGA deteriorates by increasing the answer precision; in a way that by 10^{-3}, the BGA fails to find the answer. Also the DSFL, the SFL and the DPSO have achieved a success rate of 100%. The SFL is the fastest algorithm among these three algorithms and the DPSO is the slowest. Although by increasing precision of answer, the difference in the convergence speeds of the SFL and the DSFL becomes smaller.

On HF_2 with a tolerance of 0.5, all algorithms achieve a success rate of 100%. Nevertheless, the performances of all algorithms deteriorate by increasing precision of answer, in terms of both speed and success rate. The deterioration slop is sharper for the BGA, then for the DPSO, the SFL and the DSFL; such that with a precision of 10^{-3} the success rate is 40%, 20%, 10% and zero for the SFL, the DSFL, the DPSO and the BGA, respectively.

Table 3 The success rate of four algorithms on high dimensional test functions ($D = 20$)

Function	Algorithm	Tolerance			
		0.5	0.1	0.01	0.001
HF1	DSFL	100% (10,982)	100% (12,256)	100% (14,813)	100% (17,393)
	SFL	100% (6,064)	100% (7,838)	100% (11,226)	100% (14,829)
	DPSO	100% (32,460)	100% (37,920)	100% (43,260)	0% (0)
	BGA	60% (44,220)	30% (46,080)	0% (0)	0% (0)
HF2	DSFL	100% (3,619)	100% (7,851)	80% (8,743)	20%,(10,090)
	SFL	100% (4,378)	100% (4,401)	60% (19,592)	16% (23,540)
	DPSO	100% (7,823)	100% (10,240)	50% (10,860)	0% (0)
	BGA	100% (6,994)	86% (19,306)	0% (0)	0% (0)
HF3	DSFL	100% (13,594)	100% (16,155)	100% (19,942)	100% (24,188)
	SFL	100% (9,112)	100% (10,673)	96% (13,489)	90% (18,150)
	DPSO	100% (30,973)	100% (38,604)	40% (31,100)	40% (35,676)
	BGA	0% (0)	0% (0)	0% (0)	0% (0)
HF4	DSFL	100% (27,059)	100% (33,016)	100% (42,154)	80% (48,952)
	SFL	100% (26,226)	100% (31,855)	80% (42,350)	20% (48,824)
	DPSO	0% (0)	0% (0)	0% (0)	0% (0)
	BGA	0% (0)	0% (0)	0% (0)	0% (0)
HF5	DSFL	100% (4,834)	100% (16,208)	100% (18,744)	100% (20,316)
	SFL	100% (9,102)	100% (11,355)	100% (15,663)	100% (21,181)
	DPSO	100% (32,700)	100% (36,376)	100% (44,892)	100% (47,980)
	BGA	0% (0)	0% (0)	0% (0)	0% (0)

The numbers in parentheses are the average function evaluations over successful runs

On HF_3, the BGA fails to find the answer and the other three algorithms have a success rate of 100% with any precision of answer; increasing the precision only increases the number of average function evaluations. Therefore, if we decrease the maximum number of function evaluations, the success rates will decrease.

On HF_4, both the BGA and the DPSO fail to find the answer; and the SFL and the DSFL have a success rate of 100% with any precision. Again, we notice that for larger tolerances, the SFL is much faster than the DSFL, but for smaller tolerances, the difference in their speed becomes negligible; even the DSFL becomes slightly faster than the SFL. This phenomenon is observed also in the simulations with higher dimensions of the search space.

On HF_5, all algorithms have achieved a success rate of 100% with all values of tolerance excluding the BGA, which fails to find the answer when the tolerance is 10^{-3}. This total failure of the BGA could be attributed to the limit of allowed function evaluations in each run, which was set to 50,000.

Table 3 presents the obtained results for number of variables equal to 20. On HF_1, HF_3 and HF_5 functions, both the DSFL and the SFL have a success rate of 100%, regardless of tolerance value. Of course by decreasing the tolerance, the average number of function evaluations increases. On HF_2 and HF_4, the success rates of these two algorithms decrease by increasing the precision of answer and the slop of deterioration for the SFL is sharper than that for the DSFL. Thus for smaller tolerances the DSFL has a higher success rate.

Table 4 The success rate of four algorithms on high dimensional test functions ($D = 50$)

Function	Algorithm	Tolerance			
		0.5	0.1	0.01	0.001
HF1	DSFL	100% (9,931)	100% (14,204)	100% (16,764)	100% (18,032)
	SFL	100% (8,641)	100% (8,761)	100% (12,691)	100% (13,450)
	DPSO	100% (32,700)	100% (36,960)	100% (44,280)	40% (47,820)
	BGA	10% (48,840)	0% (0)	0% (0)	0% (0)
HF2	DSFL	100% (6,272)	80% (6,957)	30% (8,743)	20% (30,090)
	SFL	100% (2,335)	100% (3,475)	20% (1,259)	20% (13,239)
	DPSO	100% (7,800)	100% (8,220)	50% (12,900)	10% (12,900)
	BGA	100% (20,460)	90% (21,300)	0% (0)	0% (0)
HF3	DSFL	100% (13,656)	100% (16,401)	100% (20,547)	100% (24,612)
	SFL	100% (9,399)	100% (10,275)	96% (16,350)	96% (23,427)
	DPSO	100% (36,372)	100% (44,104)	20% (47,340)	0% (0)
	BGA	0% (0)	0% (0)	0% (0)	0% (0)
HF4	DSFL	100% (30,511)	100% (36,005)	100% (46,028)	80% (50,000)
	SFL	100% (29,755)	100% (37,629)	60% (44,974)	20% (42,500)
	DPSO	0% (0)	0% (0)	0% (0)	0% (0)
	BGA	0% (0)	0% (0)	0% (0)	0% (0)
HF5	DSFL	100% (15,788)	100% (17,136)	100%,(19,810)	100% (21,091)
	SFL	100% (13,234)	100% (16,510)	100% (16,571)	100% (22,492)
	DPSO	100% (35,268)	100% (38,436)	80% (44,640)	60% (46,640)
	BGA	0% (0)	0% (0)	0% (0)	0% (0)

The numbers in parentheses are the average function evaluations over successful runs

On HF_1 and HF_2, the DPSO fails to find the answer when tolerance of answer is 10^{-3} or smaller. On HF_3 its success rate decreases to 40% when the tolerance value is 10^{-2} or smaller. On HF_4, it fails to find answer with any value of tolerance. And on HF_5, increasing the precision only increases the average number of function evaluations. Since with 10^{-3}, its average function evaluations have approached the limit value of 50,000, it could be predicted that by increasing the precision beyond 10^{-3}, the success rate of the algorithm will deteriorate.

The BGA does not show good performance on any function. Its best performance is obtained on HF_1 (60%) with a tolerance of 0.5, while the average number of function evaluations is close to the limit (50,000). The BGA fails on HF_3, HF_4 and HF_5, totally, and its success rate on HF_2 is 30% only when the tolerance is 0.5, and for smaller values of tolerance, it fails totally.

Finally, Tables 4 and 5 demonstrate the obtained results for number of variables equal to 50 and 100 respectively. The DSFL, the SFL and the DPSO have achieved a success rate of 100% on HF_1, HF_3 and HF_5 functions, with every value of tolerance, for both $D = 50$ and $D = 100$. On HF_2, their success rates are almost equal. Although with larger tolerances the SFL is faster than the DSFL, with smaller tolerances, the speed of the DSFL approaches the speed of the SFL. For $D = 50$, on HF_2 and HF_5 the DSFL is even slightly faster than the SFL. For $D = 100$ and with a tolerance of 10^{-3}, the DSFL is slightly faster than the SFL on HF_3 and HF_5.

Table 5 The success rate of four algorithms on high dimensional test functions ($D = 100$)

Function	Algorithm	Tolerance			
		0.5	0.1	0.01	0.001
HF1	DSFL	100% (12,931)	100% (14,211)	100% (17,048)	100% (18,963)
	SFL	100% (7,324)	100% (9,644)	100% (13,649)	100% (16,310)
	DPSO	100% (31,080)	100% (34,680)	100% (41,880)	40% (42,060)
	BGA	0% (0)	0% (0)	0% (0)	0% (0)
HF2	DSFL	100% (5,627)	100% (7,435)	60% (8,569)	20% (27,680)
	SFL	100% (1,972)	100% (2,980)	60% (8,588)	20% (16,279)
	DPSO	100% (8,260)	100% (13,020)	20% (17,690)	0% (0)
	BGA	80% (19,560)	90% (21,300)	10% (23,540)	0% (0)
HF3	DSFL	100% (13,795)	100% (16,281)	100% (20,289)	100% (24,195)
	SFL	100% (11,242)	100% (13,954)	96% (22,456)	96% (27,646)
	DPSO	100% (37,836)	60% (41,600)	0% (0)	0% (0)
	BGA	0% (0)	0% (0)	0% (0)	0% (0)
HF4	DSFL	100% (33,943)	100% (35,325)	100% (48,362)	60% (49,879)
	SFL	100% (34,761)	100% (39,020)	60% (46,352)	20% (49,665)
	DPSO	0% (0)	0% (0)	0% (0)	0% (0)
	BGA	0% (0)	0% (0)	0% (0)	0% (0)
HF5	DSFL	100% (15,437)	100% (18,616)	100% (19,897)	100% (22,675)
	SFL	100% (12,562)	100% (17,546)	100% (18,112)	100% (23,117)
	DPSO	100% (37,308)	60% (43,580)	40% (43,380)	20% (47,940)
	BGA	0% (0)	0% (0)	0% (0)	0% (0)

The numbers in parentheses are the average function evaluations over successful runs

For $D = 50$ and $D = 100$ with a tolerance of 0.5, the DPSO achieves a success rate of 100% on HF_1, HF_2, HF_3 and HF_5, but in most of the cases it is at least twice slower than the SFL. For $D = 50$ with a tolerance of 10^{-3}, the success rate of the DPSO is 40%, 10%, 0%, 0% and 0% on HF_1 to HF_5, respectively, which are much less than the success rate of the DSFL on these functions, i.e. 100%, 20%, 100%, 80% and 100%, respectively. For $D = 100$ with a tolerance of 10^{-3}, the success rates of the DSFL on HF_1 to HF_5 are 100%, 20%, 100%, 60% and 100%, respectively, while the success rates of the DPSO are only 40%, 0%, 0%, 0% and 0%. And this comparison again demonstrates the supremacy of the DSFL over the DPSO.

On HF_4, for $D = 50$ and 100, with a tolerance of 10^{-3}, the DSFL's success rates are four and three times better than those of the SFL, respectively. Excluding the HF_2 while the tolerance is 0.1 or larger, the performance of the BGA is almost disastrous for both $D = 50$ and 100.

7 General conclusions

In this paper, we presented a discrete version of the shuffled frog leaping algorithm and compared its performance with the SFL, the DPSO and the BGA algorithms on 12 difficult nonlinear and multimodal functions in two states of low and high dimensions. Our obser-

vations on low dimensional and high dimensional functions are consistent. Our proposed algorithm, i.e. the DSFL, outperforms the BGA and the DPSO in terms of both success rate and speed. On low dimensional functions and for large values of tolerance the DSFL is slower than the SFL, but their success rate are equal. On high dimensional functions and for smaller values of tolerance the DSFL becomes faster than the SFL, also its success rate is better than that of the SFL.

The obtained results demonstrate that the DSFL performs very well by increasing the number of variables and the required precision of answer, in terms of both speed and success rate. For high dimensional problems, for intrinsically discrete problems, also when the required precision of the answer is high, the DSFL is the most efficient algorithm.

A Appendix Benchmark functions

A.1 The low dimensional benchmark functions

LF_1- Sphere function (Chelouah and Siarry 2005):

$$f_1(x) = x_1^2 + x_2^2 + x_3^2; \quad -5.12 < x_i < 5.12 \tag{12}$$

It has one single minimum (local and global) with $LF_1 = 0$ at $(0, 0, 0)$. It is smooth and unimodal.

LF_2- Zakharov function (Chelouah and Siarry 2005):

$$f_2(x_1, x_2, x_3) = \sum_{i=1}^{3} [x_i^2 + (0.5ix_i)^2 + (0.5ix_i)^4]; \quad -5 < x_i < 10 \tag{13}$$

It has several local minima (exact number unspecified in usual literature), and the unique global minimum with $LF_2 = 0$ at $(0, 0, 0)$.

LF_3- Branin function (Liu et al. 2005):

$$f_3(x_1, x_2) = (x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1)^2 + 10(1 - \frac{1}{8\pi})\cos x_1 + 10$$
$$-5 < x_1 < 10; \quad 0 < x_2 < 15 \tag{14}$$

The global minimum is approximately 0.398 at three points: $(-\pi, 12.275)$, $(\pi, 2.275)$ and $(9.425, 2.475)$.

LF_4- Rastrigin function (Liu et al. 2005):

$$f_4(x) = x_1^2 + x_2^2 - \cos 18x_1 - \cos 18x_2; \quad -1 < x_i < 1 \tag{15}$$

The global minimum is -2 at the point $(0, 0)$. There are about 50 local minima arranged in a lattice configuration.

LF_5- B2 function (Chelouah and Siarry 2005):

$$f_5(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$$
$$-100 < x_i < 100 \tag{16}$$

It has several local minima (exact number unspecified in usual literature), and unique global minimum with $LF_5 = 0$ at $(0, 0)$.

LF_6- Function (Lee and Yao 2004; Leung and Wang 2001; Yao et al. 1999):

$$f_6(x) = \sum_{i=1}^{5} ([x_i + 0.5])^2; \quad -100 < x_i < 100 \tag{17}$$

It has global minimum with $LF_6 = 0$ at $(0, 0, 0, 0, 0)$.

LF_7- Quartic function with Noise (Lee and Yao 2004; Leung and Wang 2001; Yao et al. 1999):

$$f_7(x_1, x_2, \ldots x_5) = \sum_{i=1}^{5} (i+1)x_i^4 + random(0, 1)$$
$$-1.28 < x_i < 1.28 \tag{18}$$

Quartic function is a simple unimodal function padded with noise. Expected fitness depends on the random noise generator (Gaussian or uniform). The random noise makes sure that the algorithm never gets the same value on the same point. But its global minimum is approximate 0 at point $(0, 0, 0, 0, 0)$.

A.2 The high dimensional benchmark functions

In all functions listed below, N is the dimension of search space.

HF_1- Griewank's function (Lee and Yao 2004; Leung and Wang 2001; Yao et al. 1999):

$$f(x) = 1 + \frac{1}{4000} \sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} \cos(x_i/\sqrt{i})$$
$$-100 < x_i < 100 \tag{19}$$

It is multimodal and non-linear and it has a global minimum with $HF_1 = 0$ at $(0, 0, \ldots, 0)$. Number of local minima for arbitrary N is unknown. By increasing the dimensions of search space, the cost function becomes flatter.

HF_2- Rosenbrock function (Lee and Yao 2004; Leung and Wang 2001; Yao et al. 1999):

$$f(x) = \sum_{i=1}^{N-1} [100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2]$$
$$-30 < x_i < 30 \tag{20}$$

Rosenbrock function is considered to be difficult, because it has a very narrow ridge. The tip of the ridge is very sharp, and it runs around a parabola. Finding the valley is a trivial task, however convergence to the global optimum is difficult and it's global minimum is HF_2 at $(1, 1, \ldots, 1)$.

HF_3- Ackley function (Lee and Yao 2004; Leung and Wang 2001; Yao et al. 1999):

$$f(x) = 20 + e - 20 \cdot e^{-0.2\left(\sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} x_i^2}\right)} - e^{\frac{1}{N} \sum_{i=1}^{N} \cos(2\pi x_i)}$$
$$-32 < x_i < 32 \tag{21}$$

It is non-linear and multimodal test function and it has a global minimum with $HF_3 = 0$ at $(0, 0, \ldots, 0)$. Number of local minima for arbitrary N is unknown.

HF_4- Extended EF10 function (Al-kazemi and Mohan 2002a):

$$f(x) = \sum_{i=1}^{N-1} (x_i^2 + x_{i+1}^2)^{0.25}[\sin^2(50(x_i^2 + x_{i+1}^2)^{0.1}) + 1]$$
$$-100 < x_i < 100 \tag{22}$$

This function is non-linear, non-separable and it has a global minimum with $HF_4 = 0$ at $(0, 0, \ldots, 0)$.

HF_5- Rastrigin's function (Lee and Yao 2004; Leung and Wang 2001; Yao et al. 1999):

$$f(x) = 10 \cdot N + \sum_{i=1}^{N} x_i^2 - 10\cos(2\pi x_i)$$
$$-5.12 < x_i < 5.12 \tag{23}$$

This function is highly multimodal and non-linear and contains millions of local minima in the interval of consideration but it has a global minimum with $HF_5 = 0$ at $(0, 0, \ldots, 0)$.

References

Afshinmanesh F, Marandi A, Rahimi-Kian R (2005) A novel binary particle swarm optimization method using artificial immune system. In: IEEE international conference on computer as a tool, pp 217–220

Al-kazemi B (2002) Multi-phase particle swarm optimization. Dissertation, University of Syracuse

Al-kazemi B, Mohan CK (2002a) Training feed forward neural networks using multi-phase particle swarm optimization. In: Proceeding of the IEEE international conference on neural information processing, pp 2615–2619

Al-kazemi B, Mohan CK (2002b) Multi-phase discrete particle swarm optimization. In: proceeding of international workshop on frontiers on evolutionary algorithms. Research Triangle Park, pp 622–625

Al-Tabtabai H, Alex PA (1999) Using genetic algorithms to solve optimization problems in construction Engineering. Constr Archit Manag 6(2):121–132

Chelouah R, Siarry P (2005) A hybrid method combining continuous tabu search and Nelder–Mead simples algorithms for the global optimization of multi minima functions. Eur J Oper Res 161:636–654

Clerc M, Kennedy J (2002) The particle swarm-explosion, stability, and convergence in a multi-dimensional complex space. IEEE Trans Evol Comput 6(1):58–73

Eberhart RC, Kennedy J (1997) A discrete binary version of the particle swarm algorithm. In: Proceeding of the IEEE international conference on systemics, cybernetics and informatics, pp 4104–4109

Eberhart RC, Shi Y (2001) Particle swarm optimization: developments, applications and resources. In: Proceeding of the 2001 congress on evolutionary computation, pp 81–86

Elbeltagi E, Hegazy T, Grierson D (2005) Comparison among five evolutionary-based optimization algorithms. Adv Eng Inf 19:43–53

Eusuff MM, Lansey KE (2003) Optimization of water distribution network design using the shuffled frog leaping algorithm. J Water Resour Plan Manag 129(3):210–225

Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley Longman, Boston

Grierson DE, Khajehpour S (2002) Method for conceptual design applied to office buildings. Comput Civil Eng 16(2):83–103

Haupt RL, Haupt SE (2004) Practical genetic algorithms. Wiley, New York

Hegazy T (1999) Optimization of construction time-cost trade-off analysis using genetic algorithms. Can J civil Eng 26(6):685–697

Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press; Extended new Edition, MIT Press, Cambridge, 1992

Joglekar A, Tungare M (2003) Genetic algorithms and their use in the design of evolvable hardware. http://www.manastungare.com/articles/genetic/genetic-algorithms.pdf

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceeding of the IEEE international conference on neural networks, pp 1942–1948

Kennedy J, Spears WM (1998) Matching algorithms to problems: an experimental test of the particle swarm and genetic algorithms on multi-modal problem generator. In: Proceeding of the IEEE international conference on evolutionary computation, pp 78–83

Lee CY, Yao X (2004) Evolutionary programming using mutations based on the levy probability distribution. IEEE Trans Evol Comput 8(1):1–13

Leung YW, Wang Y (2001) An orthogonal genetic algorithm with quantization for global numerical optimization. IEEE Trans Evol Comput 5(1):41–53

Liu B, Wang L, Jin CYH, Tang DF, Huang DX (2005) Improved particle swarm optimization combined with chaos. Chaos Solitons Fractals 25(5):1261–1271

Lovbjerg M (2002) Improving particle swarm optimization by hybridization of stochastic search heuristics and self-organized criticality. Dissertation, University of Aarhus

Sharaf AM, El-Gammal A (2009) A novel discrete multi-objective particle swarm optimization (MOPSO) of Optimal Shunt Power filter. In: IEEE conference on power systems, pp 1–7

Shi Y (2004) Particle swarm optimization. Inc. IEEE Neural Network Society, pp 8–13

Tseng CT, Liao CJ (2007) A discrete particle swarm optimization for lot-streaming flowshop scheduling problem. Eur J Oper Res 191(2):360–373

Xu Y, Chen G, Yu J (2006) Three sub-swarm discrete particle swarm optimization algorithm. In: IEEE international conference on information acquisition, pp 1224–1228

Xu Y, Wang Q, Hu J (2008) An improved discrete particle swarm optimization based on cooperative swarms. In: IEEE international conference on web intelligence and intelligent agent technology, pp 79–82

Yao X, Liu Y, Lin G (1999) Evolutionary programming made faster. IEEE Trans Evol Comput 3(2):82–102

Zhang YL, Ma LH, Zhang LY, Qian JX (2003) On the convergence analysis and parameter selection in particle swarm optimization. In: International conference on machine learning and cybernetic, pp 1802–1807