Knowledge-Based Systems 75 (2015) 1–18

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Stochastic Fractal Search: A powerful metaheuristic algorithm

Hamid Salimi

School of Mathematics and Computer Science, University of Tehran, Tehran, Iran

ARTICLE INFO

Article history:
Received 30 August 2013
Received in revised form 24 June 2014
Accepted 30 July 2014
Available online 27 August 2014

Keywords:
Metaheuristic algorithms
Fractals
Random fractals
Global optimization
Exploration
Exploitation

ABSTRACT

Evolutionary Algorithms (EAs) are well-known terms in many science fields. EAs usually interfere with science problems when common mathematical methods are unable to provide a good solution or finding the exact solution requires an unreasonable amount of time. Nowadays, many EA methods have been proposed and developed. Most of them imitate natural behavior, such as swarm animal movement. In this paper, inspired by the natural phenomenon of growth, a new metaheuristic algorithm is presented that uses a mathematic concept called the fractal. Using the diffusion property which is seen regularly in random fractals, the particles in the new algorithm explore the search space more efficiently. To verify the performance of our approach, both the constrained and unconstrained standard benchmark functions are employed. Some classic functions including unimodal and multimodal functions, as well as some modern hard functions, are employed as unconstrained benchmark functions; On the other hand, some well-known engineering design optimization problems commonly used in the literature are considered as constrained benchmark functions. Numerical results and comparisons with other state of the art stochastic algorithms are also provided. Considering both convergence and accuracy simultaneously, experimental results prove that the proposed method performs significantly better than other previous well-known metaheuristic algorithms in terms of avoiding getting stuck in local minimums, and finding the global minimum.

1. Introduction

There are countless applications for optimization in our world. Nowadays, many companies have faced problems in need of optimization. Indeed, there are many challenging problems in industry and science which are really necessary to tackle. They can be formulated as optimization problems.

We need optimization to minimize time, cost, or risk, and maximize profit, quality or efficiency. Numerous complex real-life optimization problems have emerged in many scientific fields such as engineering, economics and business [1], that cannot be solved in a reasonable amount of time and meanwhile yield a precise answer. Indeed, such problems are often highly nonlinear. Moreover, many of them include many different variables and act under complex constraints. These constraints are either in the form of simple bounds such as ranges of material properties, or in the form of nonlinear relationships such as maximum stress, maximum deflection, minimum load capacity, or geometrical configuration [2]. On the other hand, since the size of the search space increases dramatically while solving high-dimensional optimization problems, classical optimization algorithms like exhaustive search do not provide a suitable solution. Therefore, the main alternative for solving this kind of problem is using approximate algorithms.

During the 1950s and 1960s, the concept of evolution was investigated by computer scientists as an optimization tool for solving engineering problems, and later on founded a technical method called approximate algorithms. Approximate algorithms can further be divided into two classes: specific heuristics and metaheuristics. The term "heuristic" originally comes from Greek and means "to discover" and "to guide an investigation" [3]. Heuristics are techniques which seek good (near optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is [4]. Specific heuristics are designed for particular problems while metaheuristics are applicable for a large variety of optimization problems, and they also accommodate to solve any optimization problem. The competence of metaheuristic algorithms can depend on the fact that they imitate the best features in nature, especially the selection of the fittest in biological systems which has evolved by natural selection over millions of years [2].

Metaheuristics are often employed to solve hard problems which need to explore a larger space. The advantage of metaheuristic algorithms is related to exploring the space efficiently without

E-mail address: salimi.hamid86@gmail.com

http://dx.doi.org/10.1016/j.knosys.2014.07.025
0950-7051/

being sensitive to the size of the search space. Typically, metaheuristics are based on three main purposes: solving problems faster, solving large problems and obtaining robust algorithms [1]. Moreover, ease of design and implementation along with flexibility should be the other features of these algorithms. Two important characteristics of metaheuristics are: intensification (or exploitation) and diversification (or exploration). Searching around the current best solutions, and selecting the best candidates or solutions are of the intensification properties, while diversification investigates the efficiency of the algorithm in exploring the search space often using the randomization method.

The last two decades have seen enormous development in use of metaheuristics in many science fields including artificial intelligence, computational intelligence, soft computing, mathematical programming, and operations research. Most metaheuristic algorithms are inspired by natural phenomena behaviors. Among them, Genetic Algorithm (GA) [5] established Darwin theory as one of the popular algorithms that mimic the natural evolution process. Particle Swarm Optimization (PSO) proposed by Eberhart and Kennedy [6], was inspired by social behavior of flocks of birds which are searching for their food. Developed by Karaboga, Artificial Bee Colony (ABC) simulates the foraging behavior of a bee swarm [7]. Like ABC, Ant colony (AC) is another optimization algorithm inspired by the foraging behavior of ant colonies [8]. "Every particle in the universe attracts every other particles", this is Newtonian gravity rule which Gravitational Search Algorithm (GSA) is based on [9]. Cuckoo Search (CS) [2] is another successful metaheuristic which mimics the cuckoo behavior reproduction strategy. Over the last decades, uses of metaheuristic algorithms have increased [10–17]. These algorithms are used to solve complex computational optimization problems, however, fast convergence along with accuracy is not guaranteed.

The aim of the presented work is to develop an optimization algorithm that overcomes the above shortcomings. In this paper, two novel metaheuristic algorithms based on fractal properties are presented which satisfy both fast convergence and accuracy in a few generations. The main contribution of this paper is the presentation of a new algorithm with new insight into solving optimization problems based on diffusion property turned up in fractals. These algorithms are able to achieve a solution that has the least (or at most, a small) error compared with the globally optimum solution within a minimal number of iterations, thus offering an improvement in terms of accuracy, convergence time and simplicity of operations.

In the first algorithm, each particle in the system tries to simulate the branching property of a dielectric breakdown, thus making it suitable as a search tool for solving global optimization problems. The second algorithm is the developed version of the first algorithm which can cover all disadvantages of the first algorithm. All procedures in the second algorithm can be divided into two processes called Diffusing and Updating processes. In the first process, to increase the chance of finding the global minimum, similar to the first algorithm, each particle diffuses around its current position. In the latter process, to explore the problem space efficiently, the second algorithm uses some random methods as updating processes. Since the second algorithm is more promising than the first algorithm, further experiments have been done to show the superiority of this algorithm. The main purpose of this study is to introduce a new method for tackling a variety of optimization problems with a novel perspective setting a sight for future researches. On the other hand, the algorithm is extended to solve other problems such as constrained optimization problems.

Preliminary studies show that this algorithm is very promising and could outperform existing algorithms such as PSO, CS, MCS, BSA, CME-ES, DE, GSA, ABC and other well-known metaheuristic algorithms.

The rest of paper is organized as follow: Section 2 summarizes Fractals and fractal properties. Fractal Search and Stochastic Fractal Search are described in Sections 3 and 4 respectively. Experimental results are demonstrated in Section 5, and our conclusion is made in Section 6.

2. Fractals

The property of an object or quantity which explains self-similarity on all scales, in a somewhat technical sense, is called fractal. The term of "fractal" comes from the Latin word *frāctus* which means "broken" or "fractured", and it was first used by Benoît Mandelbrot in 1975. Mandelbrot also tried to use the concept of fractal theories to describe geometric patterns in nature [18].

Developing research in this area, the example list of fractals including structures from microscopic aggregates to the cluster of galaxies has been become very long. Far-from-equilibrium growth phenomena are an important field where fractals observe, and are engaged to many fields of science and technology. Some examples for such processes include dendritic solidification in an undercooled medium, viscous fingering which is observed when a viscous fluid is injected into a more viscous one, and electrodepositing of ions onto an electrode [19].

Typically, to generate a fractal shape, some common methods such as: Iterated function systems [20], Strange attractors [21], L-systems [22], Finite subdivision rules [23] and Random fractals [24] are used. Based on the fractal characteristics, our new metaheuristic method inspires random fractals grown by Diffusion Limited Aggregation (DLA) method concept as a successful search algorithm in both accuracy and time consumption.

2.1. Random fractals

Random fractals can be generated by modifying the iteration process via stochastic rules such as Levy flight, Gaussian walks, percolation clusters, self-avoiding walks, fractal landscapes, trajectories of Brownian motion and the Brownian tree (i.e., dendritic fractals generated by modeling diffusion-limited aggregation or reaction-limited aggregation clusters) [19]. Some random fractals, such as the clusters describing a bacterial colony, can be generated by a physically motivated model called "Diffusion Limited Aggregation" (DLA) [25]. For simplicity, consider the formation of such a cluster on a plane, with the initial (seed) particle located at the origin. Other particles are then generated randomly around the original point, and cause diffusion. To simulate the diffusion process, a mathematical algorithm like random walk has been employed. The diffusing particle sticks to the seed particle which is made from it. This process is repeated until a cluster has formed. While forming the cluster, the probability of particle stuck to the end has increased comparing to the ones that penetrate the interior. Therefore, this property leads a cluster to the branch-like structure (Fig. 1).

2.2. Dielectric breakdown

Narrow discharge branchings which are frequently seen in nature are called dielectric breakdown. Study on dielectric breakdown properties shows that the branching tendency can be modeled into complicated stochastic patterns. Examples are lightning, surface discharges (Lichtenberg figures), and treeing in polymers. The global structure of branched discharges often shows a close structural similarity within a large class of discharge types but at the moment even a qualitative classification of these structures is missing. Niemeyer et al. [26] showed that branched discharges follow fractal properties, and proposed a new stochastic

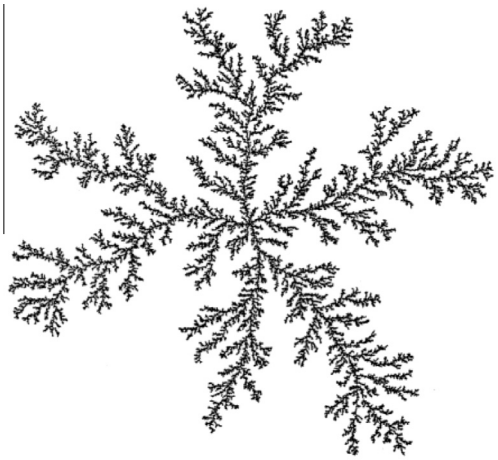


Fig. 1. A simple fractal growth by DLA method.

model to describe the discharge pattern of a dielectric breakdown. Their model is roughly similar to DLA.

3. Fractal search

According to the previous section, our first method has employed both fractal growth (DLA method) and potential theory. Loosely speaking, Fractal Search uses three simple rules to find a solution:

1. Each particle has an electrical potential energy.
2. Each particle diffuses, and causes some other random particles to be created, and the energy of the seed particle is divided among generated particles.
3. Only few of the best particles remain in each generation, and the rest of the particles are disregarded.

Suppose P ($where$ $1 \leqslant P \leqslant 20$) particles are taken into account to find the solution for a problem. At the beginning, each particle P_i has been placed randomly in the search space with equal energy E_i obtained from Eq. (1).

$$E_i = \frac{E}{P} \tag{1}$$

where E, is the maximum electric potential energy considered to solve a problem. To run fractal optimizer, each particle is diffused in each generation, and creates some other particles based on levy flight. A levy flight is a particular type of Brownian motion involving a multilevel of random steps in which, every now and again, the randomly staggering object takes a flying leap into another region of space. A levy flight can be also used to model the spread of an epidemic [24]. In this model, we apply levy flight to model DLA growth. The following equation describes the levy distribution:

$$L(x) = \frac{1}{\pi}\int_0^{\infty} \exp(-\alpha q^{\beta})\cos(qx)dx \tag{2}$$

where β is the distribution index which is restricted to $0 < \beta \leqslant 2$, and α is the distribution scale factor.

Demonstrated in Fig. 2, particle diffusion causes the creation of new particles with random different positions around the particle.

As a result of diffusion, some other particles are created. To create each point in the diffusion process, both Levy flight and Gaussian distributions are applied based on Eqs. (3) and (4), respectively.

$$x_i^q = x_i + \alpha_i^q \otimes Levy(\lambda), \quad where \quad q \quad \text{is an index}(1 \leqslant q \leqslant \text{Maximum Diffusion Number}) \tag{3}$$

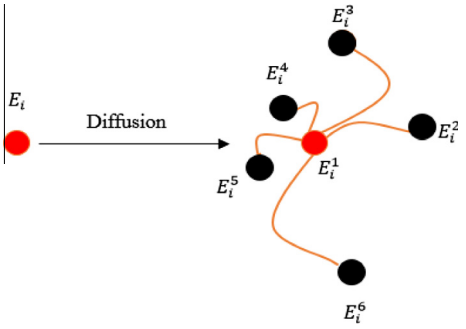


Fig. 2. Diffusing a particle.

$$x_i^q = x_i + \beta \times \text{Gaussian}(P_i, |BP|) - (\gamma \times BP - \gamma' \times P_i), \tag{4}$$

where q is the number of particles acquired from the diffusion of the main particle. The product $\otimes$ means entry-wise multiplications. β in Eq. (4) is equal to $\frac{\log(g)}{g}$, where g is considered the number of iterations. Gaussian(P_i,$|BP|$) is the Gaussian distribution in which the mean and standard deviation are P_i and BP respectively. BP is mentioned as the position of the best point. Furthermore, γ and γ' are random parameters between 0 and 1. To use the advantage of both Levy and Gaussian walks, fractal search algorithm swaps randomly between them. That is, because Levy distribution usually is employed to guaranty a fast converging algorithm while Gaussian walk is used to exploit a better estimation of the final result.

Since the search approach relies entirely on random walks, a fast convergence cannot be guaranteed. Therefore, the α parameter plays an important role in terms of fast convergence. Two equations are considered for α, one of them is used to search in a wider space, and the other is applied to find a solution with higher accuracy. Two equations used for α are as follow:

$$\alpha_i = \frac{\log(\min(\hat{E})) \times (U - D)}{g \times \log(E_i)} \tag{5}$$

$$\alpha_i = \frac{(U - D)}{(g \times \log(E_i))^{\varepsilon}} \tag{6}$$

where $\min(\hat{E})$ is consider as the minimum energy which a particle has in the whole system. U and D are the upper and lower bound of the search space. g is the generation number. E_i is the particle energy of P_i, and ε is considered as the power which is usually fixed by $\frac{3}{2}$.

After diffusing the particle, the main problem is how to distribute the particle energy among other created particles. A simple idea is presented to distribute energy, and that is: the better the obtained fitness particle, the greater the energy. Let q be the number of particles generated by diffusing the particle P_i with energy E_i. Each diffused particle has a fitness value of f_j where $j = 1, 2, \ldots, q$. The distribution energy equation can be defined as follows:

$$E_i^j = \left[\left(\frac{f_j}{f_i + \sum_{k=1}^{q} f_k}\right)\right] \times E_i \tag{7}$$

where f_i is the fitness value of the main particle before diffusion.

Although this model performed well in both local and global searches, during iteration the search complexity increased due to the creation of new particles based on diffusion. To tackle this problem, only a few of the best particles are passed into the next generation (less than 10% of all the particles in each generation). The obtained energy from discarding the particles is consumed for the remaining particles and the creation of new particles.

Suppose ϕ is the sum of all energy obtained from omitting the particles. Let μ be the distributing rate of energy between

remained particles and created new particles. The distribution energy equation for the remaining particles is as follows:

$$E^t_{new} = E^t_{old} + \left[\left(\frac{f_t}{\sum_{k=1}^{\xi} f_k}\right) \times \phi\right] \times \mu \tag{8}$$

where E^t_{new} and E^t_{old} are the energy of particle t after and before distributing energy. On the other hand, ξ is the number of all particles in the iteration. For each particle that is going to diffuse, the number of particles considered to be created and placed randomly in the search space is calculated according to the following equation:

$$\vartheta = \frac{\log(\text{number of discarded particles})}{\log(\text{maximum diffusion})} \tag{9}$$

The energy distributed for creating each particle is equal to that of others, and is as follows:

$$E'_c = \frac{\phi \times (1-\mu)}{\vartheta}, \quad c = 1, 2, \ldots, \vartheta \tag{10}$$

The standard Fractal Search optimization algorithm can be described as follows:

4. Stochastic fractal search

Although Fractal Search performs well in finding the solution, but this approach suffers from some disadvantages. The main one is having a lot of parameters that need to be well addressed, and the other is that information exchange does not occur among particles. Exchanging information among all participating points in the group is an attempt to speed up convergence to minimum. In FS, no information exchange has turned up between individuals, and, it is essential that the search is performed independently, so, this situation is rectified in our new algorithm by adding a phase called updating process. On the other hand, since Fractal Search is a dynamic algorithm in the sense that the number of agents in the algorithm is modified, we have faced a trade-off between accuracy and time consumption. Therefore, to tackle the mentioned problems, we introduce another version of Fractal Search called Stochastic Fractal Search (SFS).

Two main processes occurred in the SFS algorithm are: The diffusing process and the updating process. In the first process, similar to Fractal Search, each particle diffuses around its current position to satisfy intensification (exploitation) property. This process increases the chance of finding the global minima, and also prevents being trapped in the local minima. In the latter process,

Fractal Search Algorithm

Initialize a population of N points with equal energy E_i
While $g <$ maximum generation or (stop criterion) **Do**
{
For each Point P_i in the system **Do**
{
Call Diffusion Process with following process:
{
q = (maximum considered number of diffusion $\times$ rand [0,1].
If Fitness Evaluation of q does not pass user error tolerance
{
Consider Eq. (5) as Levy coefficient for larger flight ($\alpha \leftarrow$ Eq. (5)).
}
Else
{
Consider Eq. (6) as Levy coefficient for small flight jump ($\alpha \leftarrow$ Eq. (6)).
}
For j = 1 to q **Do**
{
%Switching random walk between Levy and Gaussian distributions
If rand [0,1] < 0.5
{
$X'_i = x_i - \text{rand}[-1,1] \times (\text{rand}[0,1] \times \text{Best Point} - \text{rand}[0,1] \times X_i) + a \otimes \text{Levy}$
}
Else
{
$X'_i = (\text{rand}[0,1] \times \text{Best Point} - \text{rand}[0,1] \times X_i) - \textbf{Gaussian}\ (\text{mu}, \text{sigma})$.
Where $mu = X_i, \text{sigma} = \left|\frac{\log(g)}{g} \times Best\ Point\right|$.
}
}
}
Divide the energy particle of P_i among particles created from Diffusion Process based on Eq. (7).
}
Sort all particles based on goodness of fitness.
Consider δ% of best particles (usually δ = 1), and discard (100 − δ)% of particles.
Distribute discarded particle energies among the best selected particle based on Eq. (10).
}

the algorithm simulates how a point in the group updates its position based on the position of other points in the group. Unlike the diffusing phase in FS which causes a dramatic increase in the number of participating points, we consider a static diffusion process for SFS. It means that the best generated particle from the diffusing process is the only particle that is considered, and the rest of the particles are discarded. In addition to efficient exploration of the problem space, SFS uses some random methods as updating processes. In other word, updating process in SFS leads us to diversification (exploration) properties in metaheuristic algorithms.

To create new particles from the diffusion procedure, two statistical methods called Levy flight and Gaussian are investigated. Preliminary studies over taking advantage of Levy and Gaussian distributions separately show, however, that although Levy flight converges faster than Gaussian walk in a few generations; Gaussian walk is more promising than Levy flight in finding global minima. Therefore, unlike Fractal Search which uses the Levy flight distribution, Gaussian distribution is the only random walk employed in the DLA growth process of SFS. Generally, a series of Gaussian walks participating in the diffusion process have been listed in the following equations:

$$GW_1 = Gaussian(\mu_{BP}, \sigma) + (\varepsilon \times BP - \varepsilon' \times P_i) \tag{11}$$

$$GW_2 = Gaussian(\mu_P, \sigma) \tag{12}$$

where ε and ε' are uniformly distributed random numbers restricted to [0,1]. BP and P_i are denoted as the position of the best point and the ith point in the group, respectively. The first two Gaussian parameters are μ_{BP} and σ where μ_{BP} is exactly equal to $|BP|$. The two the latter parameters are μ_P and σ where μ_P is equal to $|P_i|$. With consideration of Gaussian parameters, the standard deviation is computed by Eq. (13):

$$\sigma = \left| \frac{\log(g)}{g} \times (P_i - BP) \right|. \tag{13}$$

To encourage a more localized search as individuals, and get closer to the solution, the term $\frac{\log(g)}{g}$ is used in order to decrease the size of Gaussian jumps, as the number of generation increases.

Assume a global optimization problem with dimension D is at hand. Therefore, each denoted individual considered to solve the problem has been built based on a D-dimensional vector. During the initialization process, each point is initialized randomly based on problem constrains by prescribing minimum and maximum bounds. The initialization equation of the jth point, P_j, is addressed as follows:

$$P_j = LB + \varepsilon \times (UB - LB) \tag{14}$$

where LB and UB are the lower and the upper problem constrained vectors, respectively. As stated in previous equations, ε is a uniformly distributed random number which is restricted to [0,1] continuous area. After initializing all points, the fitness function of each point is computed to attain the best point (BP) among all points. According to the exploitation property in the diffusion procedure, all points have roamed around their current position to exploit problem search space. On the other hand, two statistical procedures aimed to increase the better space exploration are considered due to the exploration property. The first statistical procedure performs on each individual vector index, and the second statistical method is then applied to all points.

For the first statistical procedure, at first, all the points are ranked based on the value of the fitness function. Each point i in the group is then given a probability value which obeys a simple uniform distribution as following equation:

$$Pa_i = \frac{rank(P_i)}{N} \tag{15}$$

where $rank(P_i)$ is consider as the rank of point P_i among the other points in the group, and N is used as the number of all points in the group. In fact, Eq. (15) wants to state that the better the point, the higher the probability. This equation is used to increase the chance of changing the position of points which have not obtained a good solution. On the other hand, the chance of passing good solutions in the next generation will increase. For each point P_i in group, based on whether or not the condition $Pa_i < \varepsilon$ is satisfied, the jth component of P_i, is updated according to Eq. (16), otherwise it remains unchanged.

$$P'_i(j) = P_r(j) - \varepsilon \times (P_t(j) - P_i(j)) \tag{16}$$

where P'_i is the new modified position of P_i. P_r and P_t are random selected points in the group, ε is the random number selected from the uniform distribution in the continuous space [0,1].

Regarding to the first statistical procedure which is carried out on the components of the points, the second statistical change is aimed to change the position of a point considering the position of other points in the group. This property improves the quality of exploration, and it satisfies the diversification property. Before starting the second procedure, once again, all points obtained from the first statistical procedure are ranked based on Eq. (15). Similar to the first statistical process, if the condition $Pa_i < \varepsilon$ is held for a new point P'_i, the current position of P'_i is modified according to Eqs. (17) and (18), otherwise no update occurs. Eqs. (17) and (18) are presented as follows:

$$P''_i = P'_i - \hat{\varepsilon} \times (P'_t - BP) \quad | \ \varepsilon' \leqslant 0.5 \tag{17}$$

$$P''_i = P'_i + \hat{\varepsilon} \times (P'_t - P'_r) \quad | \ \varepsilon' > 0.5 \tag{18}$$

where P'_r and P'_t are random selected points obtained from the first procedure, and $\hat{\varepsilon}$ are random numbers generated by the Gaussian Normal distribution. The new point P''_i is replaced by P'_i if its fitness function value is better than P'_i. You can find the algorithm flowchart in Appendix B. The standard Stochastic Fractal Search can be described as follows:

Stochastic Fractal Search Pseudo Algorithm

```
Initialize a population of N points
While g < maximum generation or (stop criterion) Do
  {
     For each Point P_i in the system Do
     {
        Call Diffusion Process with the following process:
        {
           q = (maximum considered number of
               diffusion).
           For j = 1 to q Do
           {
           If (user applies the first Gaussian walk to solve
              the problem)
           {
              Create a new point based on Eq. (11).
           }
           Else (user sets the second Gaussian Walks to
                solve the problem)
           {
              Create a new point based on Eq. (12).
           }
        }
     }
  }
```

```
Call Updating Process with the following process
{
    Fist Updating Process:
    First, all points are ranked based on Eq. (15).
    For each Point P_i in the system Do
        {
            For each component j in P_i Do
                {
                If rand[0, 1] ⩾ Pa_i
                    {
                        Update the component inbased on
                        Eq. (16).
                    }
                Else
                {
                        Do nothing.
                }
            }
        Second Updating Process:
        Once again, all points obtained by the first update
        process are ranked based on Eq. (15).
        For each new point P'_i in the system Do
            {
                If (rand[0, 1] ⩾ Pa'_i)
                {
                    Update the position based on Eqs. (17)
                    and (18).
                }
                Else
                {
                    Do nothing.
                }
            }
        }
}
```

5. Experimental result

To study our new approaches, experimental results were carried out on both constrained and unconstrained benchmark functions. For constrained benchmark functions, some engineering optimization design problems commonly used in literature were used. For unconstrained benchmark functions, two types of benchmark functions including classic and modern benchmark functions with different properties were employed.

5.1. Experiment I – Classic benchmark functions

To evaluate the relative performance of the proposed method on classic benchmark functions, a series of 23 standard benchmark functions were taken from those widely used in literature [27]. Without any modification, these functions are presented in Table 1. Typically, these functions can be divided into three categories: (1) Unimodal functions, (2) Multimodal functions and (3) Multimodal test functions with fixed dimension.

In this table, D represents the dimension of the function, and the minimum value of the function is demonstrated as Optimum. Except f_{08}, f_{10} and f_{12} which have minimum value of $(-418.9829 * n)$, 8.8818e−16 and 1.5705e−032 respectively, the rest of the minimum values of functions f_{01}–f_{13} are zero. For the functions f_{01}–f_{13}, the optimal locations occur in $[0, 0, 0, \ldots, 0]$, except f_5, f_{08}, f_{12} and f_{13} in which they taken place in $[1, 1, 1, \ldots, 1]$, $[420.96, 420.96, \ldots, 420.96]$, $[-1, -1, -1, \ldots, -1]$ and $[1, 1, 1, \ldots, 1]$ respectively. The detailed description of these functions f_{14}–f_{23}, are shown in Table 2. The following section investigates the unimodal and multimodal functions and their properties.

5.1.1. Unimodal high-dimensional functions

Unimodal high-dimensional functions are designed to test the convergence rate of search algorithms. Since there are several methods specially designed to optimize unimodal functions, the convergence rate is more important than the final results. Some kinds of these functions are f_{01} to f_{07}.

5.1.2. Multimodal high-dimensional functions

Because of the fact that we encounter many local minimums, multimodal functions are the most difficult problems to optimize. For these kinds of problems, because the algorithms have challenged to escape from poor local optima and locate a near-global optimum, the final results are more important. Some kinds of these functions are f_{08} to f_{13}.

5.1.3. Multimodal low-dimensional functions

Similar to multimodal function stated in the previous stage, f_{14} to f_{23} belong to this category but with little difference, in the sense that since the dimensions of these function are low, we have to deal with a small number of local minimums.

5.1.4. Algorithm comparison

SFS algorithm is implemented by MATLAB 2010 version 7.11 (b). You can also request FS and SFS code from the author or refer to Appendix A. To have a fair comparison, our SFS and FS algorithms have been compared with other well-known algorithms such as: PSO [6], MCS [28], GSA [9], CS [2] and ABC [7]. The maximum generation numbers used for all functions are shown in Table 3. Appendix A shows the download link of all algorithms used in our experiments.

The control parameters of the comparative algorithms are given according to Table 4. These parameters are commonly used in literature.

For Fractal Search (FS), the maximum energy considered to solve all problems was set to E = 2500 *(joule)*, and the initial population size was set to 5. The Maximum Diffusion Number (MDN) was 10, the Remaining Point (RP) was tuned to 0.01, and the Distributed Energy (DE) was considered to be 99%. The obtained results by well-known algorithms over unimodal functions (f_1–f_7) are reported in Table 5. In this table, the average results of 25 independent runs are summarized. We have also ranked the algorithms from smallest mean solution to the highest solution. At last, we have reported the average ranks obtained by algorithms, and concluded the final ranks as the overall rank. As it can be concluded from Table 5, although SFS and FS satisfy the condition of convergence rate which is important for unimodal function, they can also find the global minimum of all functions. Moreover, the reported results in Table 5 prove the superiority of SFS and FS over other successful metaheuristic algorithms in solving unimodal functions. Fig. 3 shows the significant difference between SFS and other algorithms in terms of convergence and accuracy in solving unimodal test functions. Among all mentioned functions, f_{05} and f_{06} are two of the hardest functions the global minimum of which all algorithms except SFS and FS have failed the attempt to find.

The results obtained by PSO, CS, MCS, GSA, ABC, FS and SFS algorithms over multimodal high dimensional functions are presented in Table 6. As stated before, f_{08}–f_{13} functions are multimodal functions with a lot of local minima. The aim of investigating these functions is scrutinizing the ability of algorithms to overcome the local minima test functions which increase exponentially as long as the dimension of search space increases. Table 6 shows

Table 1
The benchmark function used in our experiment.

Test function	D	Range	Optimum
$f_{01}=\sum_{i=1}^{n}x_i^2$	30	$[-100,100]$	0
$f_{02}=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_i\rvert$	30	$[-10,10]$	0
$f_{03}=\sum_{i=1}^{n}\left(\sum_{j=1}^{i}x_j\right)^2$	30	$[-100,100]$	0
$f_{04}=\max_i\{\lvert x_i\rvert,1\leqslant i\leqslant D\}$	30	$[-100,100]$	0
$f_{05}=\sum_{i=1}^{D-1}\left[100\left(x_{i+1}-x_i^2\right)^2+\left(x_i-1\right)^2\right]$	30	$[-30,30]$	0
$f_{06}=\sum_{i=1}^{D}(\lfloor x_i+0.5\rfloor)^2$	30	$[-100,100]$	0
$f_{07}=\sum_{i=1}^{D}ix_i^4+random[0,1)$	30	$[-1.28,1.28]$	0
$f_{08}=\sum_{i=1}^{D}-x_i\sin\left(\sqrt{\lvert x_i\rvert}\right)$	30	$[-500,500]$	$-418.9829*n$
$f_{09}=\sum_{i=1}^{D}\left[x_i^2-10\cos(2\pi x_i)+10\right]$	30	$[-5.12,5.12]$	0
$f_{10}=-20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}x_i^2}\right)-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos 2\pi x_i\right)+20+e$	30	$[-32,32]$	8.8818e−16
$f_{11}=\frac{1}{400}\sum_{i=1}^{D}x_i^2-\prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	30	$[-600,600]$	0
$f_{12}=\frac{\pi}{D}\{10\sin^2(\pi y_i)+\sum_{i=1}^{D-1}(y_i-1)^2[1+10\sin^2(\pi y_i+1)]+(yD-1)^2$ $+\sum_{i=1}^{D}u(x_i,10,100,4)\}$ $y_i=1+\frac{x_i+1}{4}\quad u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m & x_i>a\\ 0 & -a<x_i<a\\ k(-x_i-a)^m & x_i<-a\end{cases}$	30	$[-50,50]$	1.5705e−032
$f_{13}=0.1\{10\sin^2(\pi y_i)+\sum_{i=1}^{D-1}(y_i-1)^2[1+10\sin^2(\pi y_i+1)]+(yD-1)^2\}+\sum_{i=1}^{D}u(x_i,10,100,4)$	30	$[-50,50]$	0
$f_{14}=\left[\frac{1}{500}+\sum_{j=1}^{25}\frac{1}{j+\sum_{i=1}^{2}(x_i-a_{ij})^6}\right]^{-1}$	2	$[-65.53,65.53]$	0.998004
$f_{15}=\sum_{i=1}^{11}\left[a_i-\frac{x_1\left(b_i^2+b_ix_j\right)}{b_i^2+b_ix_3+x_4}\right]^2$	4	$[-5,5]$	0.0003075
$f_{16}=4x_1^2-2.1x_i^4+\frac{1}{3}x_1^6+x_1x_2-4x_2^2+4x_2^4$	2	$[-5,5]$	−1.0316285
$f_{17}=\left(x_2-\frac{5.1}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos x_1+10$	2	$[-5,10]*[0,15]$	0.398
$f_{18}=\left[1+(x_1+x_2+1)^2\left(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2\right)\right]$ $\times\left[30+(2x_1-3x_2)^2\left(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2\right)\right]$	2	$[-5,5]$	3
$f_{19}=-\sum_{i=1}^{4}c_i\exp\left(-\sum_{j=1}^{3}a_{ij}(x_j-p_{ij})^2\right)$	3	$[0,1]$	−3.86
$f_{20}=-\sum_{i=1}^{4}c_i\exp\left(-\sum_{j=1}^{6}a_{ij}(x_j-p_{ij})^2\right)$	6	$[0,1]$	−3.32
$f_{21}=-\sum_{i=1}^{5}\left[(X-a_i)(X-a_i)^T+c_i\right]^{-1}$	4	$[0,10]$	−10.1532
$f_{22}=-\sum_{i=1}^{7}\left[(X-a_i)(X-a_i)^T+c_i\right]^{-1}$	4	$[0,10]$	−10.4029
$f_{23}=-\sum_{i=1}^{10}\left[(X-a_i)(X-a_i)^T+c_i\right]^{-1}$	4	$[0,10]$	−10.5364

Table 2
The description detail of low-dimensional multimodal functions.

Functions	Optimal location	Optimum
f_{14}	$(-32,-32)$	0.998004
f_{15}	$(0.1928,0.1908,0.1231,0.1358)$	0.0003075
f_{16}	$(0.089,-0.712)$, $(-0.089,0.712)$	−1.0316285
f_{17}	$(-3.14,12.27)$, $(3.14,2.275)$, $(9.42,2.42)$	0.398
f_{18}	$(0,-1)$	3
f_{19}	$(0.114,0.556,0.852)$	−3.86
f_{20}	$(0.201,0.15,0.477,0.275,0.311,0.657)$	−3.32
f_{21}	5 local minima in a_{ij} $i=1,2,\ldots,5$	−10.1532
f_{22}	7 local minima in a_{ij} $i=1,2,\ldots,7$	−10.4029
f_{23}	10 local minima in a_{ij} $i=1,2,\ldots,10$	−10.5364

Table 3
Maximum iteration for each classic function.

Functions	Iterations	Functions	Iterations
f_{01}	500	f_{13}	2000
f_{02}	950	f_{14}	150
f_{03}	500	f_{15}	400
f_{04}	1000	f_{16}	200
f_{05}	8000	f_{17}	180
f_{06}	15	f_{18}	200
f_{07}	1500	f_{19}	100
f_{08}	1500	f_{20}	250
f_{09}	40	f_{21}	200
f_{10}	60	f_{22}	200
f_{11}	70	f_{23}	200
f_{12}	2000		

Table 4
The control parameters of search algorithms.

Name	Specification	Population size
PSO	$\omega=0.6$, as weight factor and $c_1=c_2=2$ [29]	100
CS	$\beta=1.5$ and $p_a=0.25$ have been recommended in [30]	100
MCS	$p_a=0.7$ and $A=100$ according to [28]	100
GSA	$G_o=100$ and $\alpha=20$, and T is set to NP [9]	100
ABC	Limit = $50D$ has been used as recommended in [7]	100
SFS	Maximum Diffusion Number (MDN) is set to 1	100

the average results of 25 independent algorithms executed over multimodal test functions. According to Table 6, SFS can succeed in getting rid of functions' local minimums, and comparing with other powerful metaheuristic algorithms, provide better results. As for the better exploration ability, SFS has the first overall rank in solving multimodal test functions. The graphical results of functions f_{08} and f_{09} are depicted in Fig. 4.

An early look to Table 6 reveals the weakness of all algorithms except SFS in preventing getting stuck in local minimums. As can be concluded from Table 6, although FS could find the global minimum of functions f_{09}, f_{10} and f_{11}, it had a low performance when solving functions like f_{08}, f_{12} and f_{13}. The poor performance of FS over the mentioned functions reflects the sensitivity of FS when initializing its parameters. On the other hand, considering the obtained results by CS and MCS along with FS which uses Levy

Table 5
Minimization result of benchmark functions $f1$–$f7$ for different algorithms.

Functions	Algorithm	PSO	CS	MCS	GSA	ABC	FS	SFS
f_{01}	Mean	7.8157	90.6072	1.0082	5.8000E−18	3.5678E−05	0	**0**
	StDev	2.1173	26.2332	0.2719	1.4422E−18	3.1143E−05	0	**0**
	Rank	6	7	5	3	4	**1**	**1**
f_{02}	Mean	0.9676	9.7033	0.1814	1.0208E−08	7.7417E−06	0	**0**
	StDev	0.0781	1.9805	0.0331	1.3353E−09	2.3977E−06	0	**0**
	Rank	6	7	5	3	4	**1**	**1**
f_{03}	Mean	65.5158	3.8434E+03	461.9995	248.3737	1.6214E+04	0	**0**
	StDev	16.4385	723.9758	122.9394	92.9878	2.2517E+03	0	**0**
	Rank	3	6	5	4	7	1	**1**
f_{04}	Mean	0.9874	7.2334	1.7347	1.0193E−09	22.1724	0	**0**
	StDev	0.0176	0.6757	0.5124	1.5271E−10	2.4676	0	**0**
	Rank	4	6	5	3	7	1	**1**
f_{05}	Mean	24.1219	4.9820	58.1470	18.8115	0.0607	15.5063	**0**
	StDev	22.9452	1.7464	33.0726	0.2069	0.1147	0.1294	**0**
	Rank	5	3	7	6	2	4	**1**
f_{06}	Mean	2.6550E+04	4.3129E+04	4.4378e+03	5.5671E+04	4.8813E+04	0	**0**
	StDev	1.6092E+03	7.2109E+03	751.8954	4.4908E+03	6.7554E+03	0	**0**
	Rank	4	5	3	7	6	**1**	**1**
f_{07}	Mean	111.8227	0.0246	0.0091	0.0048	0.0621	5.7231E−04	**3.0566e−004**
	StDev	18.2958	0.0079	0.0022	0.0022	0.0131	1.4647e−005	**4.3471E−04**
	Rank	7	5	4	3	6	2	**1**
Average rank		5	5.5714	4.8571	4.1429	5.1429	1.5714	**1**
Overall rank		5	7	4	3	6	2	**1**

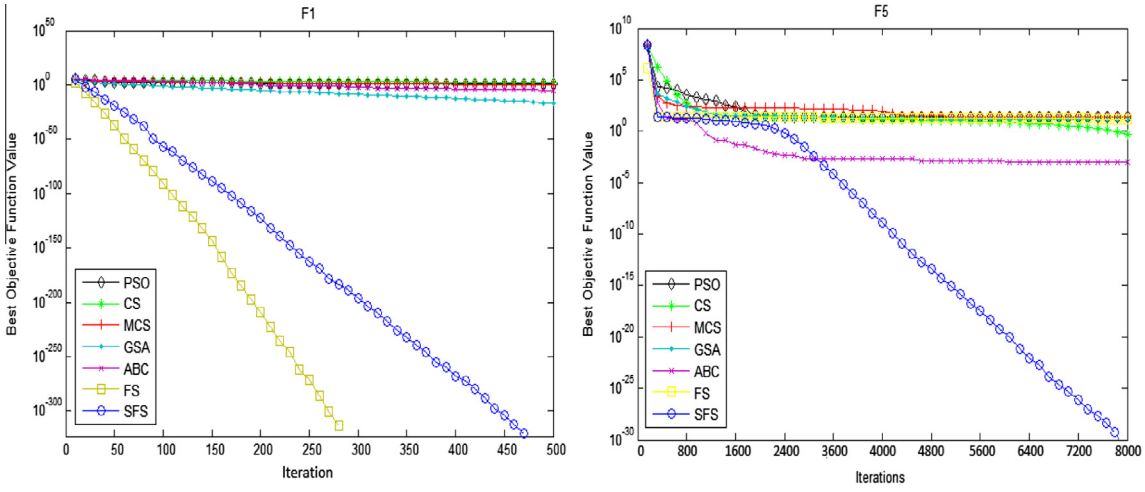


Fig. 3. Graphical results of obtained results of PSO, CS, MCS, GSA, ABC, FS and SFS over function f1 and f5.

flight distribution, the disadvantage of Levy flight in getting stuck to local minimums can be easily concluded.

For functions f_{14}–f_{23} with few local minima due to low dimensions, the final results are reported in Table 7. These functions are similar to the multimodal function with a small difference. The main difference of these functions with functions of f_{08}–f_{13} is that instead of containing high dimension functions, these functions deal with low and fixed dimensions. Therefore, it is hard to have a fair comparison among algorithms. The summarized results of Table 7 show that in the case of some functions like f_{14}, f_{16}, f_{17}, f_{18}, f_{19} and so on, almost all metaheuristic algorithms could reach optimal or near optimal solutions. However, we still ranked these algorithms. As can be seen from Table 7, SFS algorithm outperforms other algorithms in solving low-dimensional multimodal functions. We also depict the graphical analysis of the ANOVA test for functions f_{14} and f_{16} in Fig. 5.

According to Table 7, FS does not provide a good solution for low-dimensional multimodal. However, SFS could overcome all low-dimensional multimodal functions, some other algorithms such as ABC and GSA also could obtain acceptable results over some low-dimensional multimodal functions such as f_{16}, f_{17}, f_{18}, f_{20}.

5.2. Experiment II – Modern benchmark functions (CEC 2010)

We also carried our experiments out on the CEC'2010 benchmark functions designed for the special issue as a competition on Large Scale Global Optimization problems [31]. The advantage of this benchmark suite is that the degree of non-separability is well defined and is adjustable. This benchmark function is known as separable problems, and includes five major categories which are as follows:

Table 6
Minimization result of benchmark functions $f08$–$f13$ for different algorithms.

Functions	Algorithm	PSO	CS	MCS	GSA	ABC	FS	SFS
f_{08}	Mean	−2.7722E+03	−8.9784E+03	−9.8013e+03	−2.9244E+03	−1.2255E+04	−6.6156e+03	**−1.2569E+04**
	StDev	424.8212	198.0568	531.0334	427.6096	57.9874	673.2601	**8.5748E−13**
	Rank	7	4	3	6	2	5	**1**
f_{09}	Mean	254.8251	294.0493	135.0405	197.3704	197.9254	0	**0**
	StDev	10.6395	14.3249	21.6173	19.4655	16.5509	0	**0**
	Rank	6	7	3	4	5	**1**	**1**
f_{10}	Mean	7.4085	19.3448	12.1188	7.3317	19.0133	8.8818E−16	**8.8818E−16**
	StDev	0.3121	0.3503	0.7515	0.6787	0.4813	0	**0**
	Rank	4	7	5	3	6	**1**	**1**
f_{11}	Mean	327.2664	212.4145	8.3201	479.4395	135.9647	0	**0**
	StDev	32.8165	39.7384	1.5369	44.4233	34.1362	0	**0**
	Rank	6	5	3	7	4	**1**	**1**
f_{12}	Mean	0.8562	1.4726	0.1379	2.8705E−20	1.7128E−13	0.1278	**1.5705E−32**
	StDev	0.0928	0.3609	0.2862	6.0309E−21	1.5969E−13	0.0418	**2.8850E−48**
	Rank	6	7	5	2	3	4	**1**
f_{13}	Mean	0.9136	0.9328	0.0039	3.9379E−19	3.3132E−12	0.8017	**1.3498E−32**
	StDev	0.0521	0.2126	0.0053	8.4161E−20	2.5622E−12	0.1226	**2.8850E−48**
	Rank	6	7	4	2	3	5	**1**
Average rank		5.8333	6.1667	3.8333	4	3.8333	2.8333	**1**
Overall rank		6	7	3	5	3	2	**1**

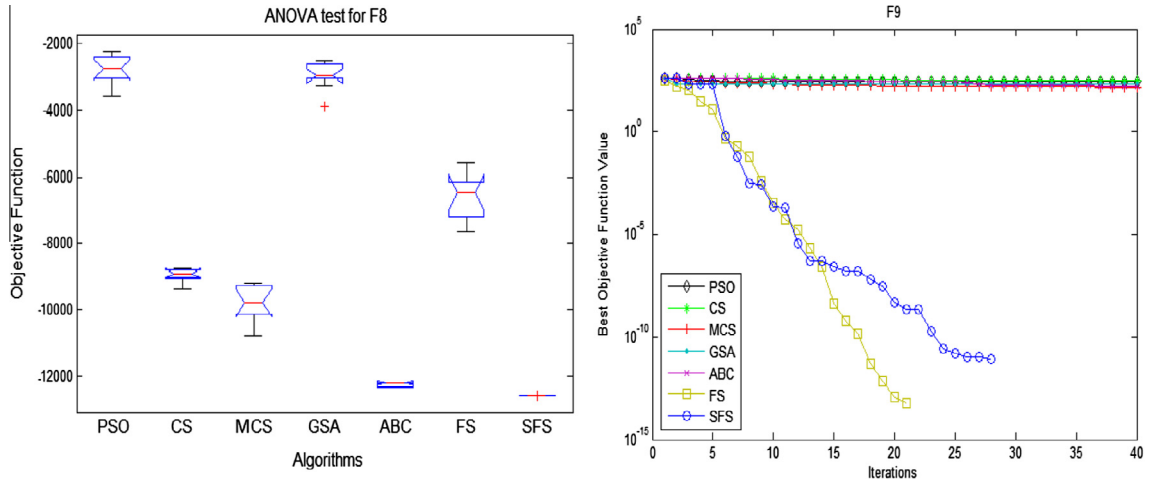


Fig. 4. ANOVA test and graphical results of functions f8 and f9 respectively.

(1) Separable Functions (f_{c1}–f_{c3})
(2) Single-group m-nonseparable Functions (f_{c4}–f_{c8})

- f_{c4}: Single-group Shifted and m-rotated Elliptic Function.
- f_{c5}: Single-group Shifted and m-rotated Rastrigin's Function.
- f_{c6}: Single-group Shifted and m-rotated Ackley's Function.
- f_{c7}: Single-group Shifted and m-rotated Schwefel's Problem 1.2.
- f_{c8}: Single-group Shifted and m-rotated Rosenbrock's Function.

(3) $\frac{m}{2n}$ – group m-nonseparable Functions (f_{c9}–f_{c13})
(4) $\frac{m}{n}$ – group m-nonseparable Functions (f_{c14}–f_{c18})
(5) Nonseparable Functions (f_{c19}–f_{c20})

In the above list, m refers to the number of interacting variables in a non-separable group and n is the dimensionality of the problem. To visit the details of the functions, you can refer to [31]. Conserving the originality of functions in the CEC2010 benchmark, we consider these benchmark functions to make them suitable for dimension 30. The maximum iterations for evaluating CEC 2010 benchmark functions are shown in Table 8.

To evaluate CEC 2010 benchmark functions, the results of SFS are compared with state-of-the art algorithms in relevant literature such as: BSA [32], CMA-ES [33,34], DE [34], ABC [7,35] and AMO [36]. The control parameters of the mentioned algorithms are stated in Table 9. These tuned parameters are commonly used in literature.

The results of 25 independent runs on CEC 2010 benchmark functions are summarized in Table 10. Like classic benchmark function, all algorithms are ranked from smallest mean solution to higher solutions. The ranks are then averaged based on these twenty functions and the average rank is obtained.

According to obtained results from Table 10, it is easily obvious that stochastic Fractal Search performs much better than other algorithms. As it can be concluded from Table 10, SFS can find the global optimum solution of most functions. As mentioned above, CEC 2010 benchmark functions can be divided into five groups. For the first group which is also called separable functions, SFS, AMO, ABC and DE can almost find the solutions comparing to the rest. The obtained results on the second group of functions called single-group m-nonseparable functions show that SFS is

Table 7
Minimization result of benchmark functions $f14$–$f23$ for different algorithms.

Functions	Algorithm	PSO	CS	MCS	GSA	ABC	FS	SFS
f_{14}	Mean	1.3959	0.9980	2.3440	5.7804	0.9980	1.8899	**0.9980**
	StDev	0.5131	1.4658E−06	2.1186	2.2194	8.5998E−16	1.2743	**0**
	Rank	4	3	6	7	2	5	**1**
f_{15}	Mean	0.0677	5.5634E−04	5.4872E−04	0.0035	8.2873E−04	0.0103	**3.0749E−04**
	StDev	0.0383	7.4905E−05	1.5447E−04	9.8772E−04	2.8229E−04	0.0131	**1.0222E−19**
	Rank	7	3	2	5	4	6	**1**
f_{16}	Mean	−1.0313	−1.0316	−1.0316	−1.0316	−1.0316	−1.0316	**−1.0316**
	StDev	4.1210E−04	7.5569E−12	3.6546E−10	1.9582E−16	2.2204E−16	8.2348E−07	**0**
	Rank	7	4	5	2	3	6	**1**
f_{17}	Mean	0.3984	0.3979	0.3979	0.3979	0.3979	0.3979	**0.3979**
	StDev	0.0010	1.1217E−08	3.6974E−09	0	2.5796E−13	4.2354E−07	**0**
	Rank	7	5	4	1	3	6	**1**
f_{18}	Mean	4.2797	3.0000	3.0000	3.0000	3.0000	3.0000	**3.0000**
	StDev	2.3313	2.9873E−09	7.8490E−08	3.2667E−15	1.0726E−08	9.0515E−06	**4.6811E−16**
	Rank	7	3	5	2	4	6	**1**
f_{19}	Mean	−3.5427	−3.8628	−3.8628	−3.8564	−3.8628	−3.8365	**−3.8628**
	StDev	0.2142	8.6922E−06	9.7121E−08	0.0038	1.7901E−10	0.0231	**3.9165E−16**
	Rank	7	4	3	5	2	6	**1**
f_{20}	Mean	−2.0641	−3.3213	−3.2744	−3.3220	−3.3220	−3.2992	**−3.3220**
	StDev	0.4120	4.8747E−04	0.0614	4.4409E−16	1.8245E−09	0.0481	**3.6260E−16**
	Rank	7	4	6	2	3	5	**1**
f_{21}	Mean	−0.7021	−10.1430	−3.4299	−7.4383	−10.1532	−5.5482	**−10.1532**
	StDev	0.2546	0.0132	2.3623	3.5926	1.5163E−07	1.5712	**1.7764E−15**
	Rank	7	3	6	4	2	5	**1**
f_{22}	Mean	−8.1481	−10.3910	−8.3021	−10.4029	−10.4029	−5.9766	**−10.4029**
	StDev	1.7271	0.0082	3.3930	2.3685E−15	1.3527E−05	2.2910	**1.6748E−15**
	Rank	6	4	5	2	3	7	**1**
f_{23}	Mean	−9.5725	−10.4927	−5.9976	−10.5364	−10.5364	−5.9830	**−10.5364**
	StDev	1.0623	0.0293	3.9158	2.1349E−15	3.2599E−05	1.8299	**1.8724E−15**
	Rank	7	4	5	2	3	6	**1**
Average rank		6.6000	3.7000	4.7000	3.2000	2.9000	5.8000	**1**
Overall rank		7	4	5	3	2	6	**1**

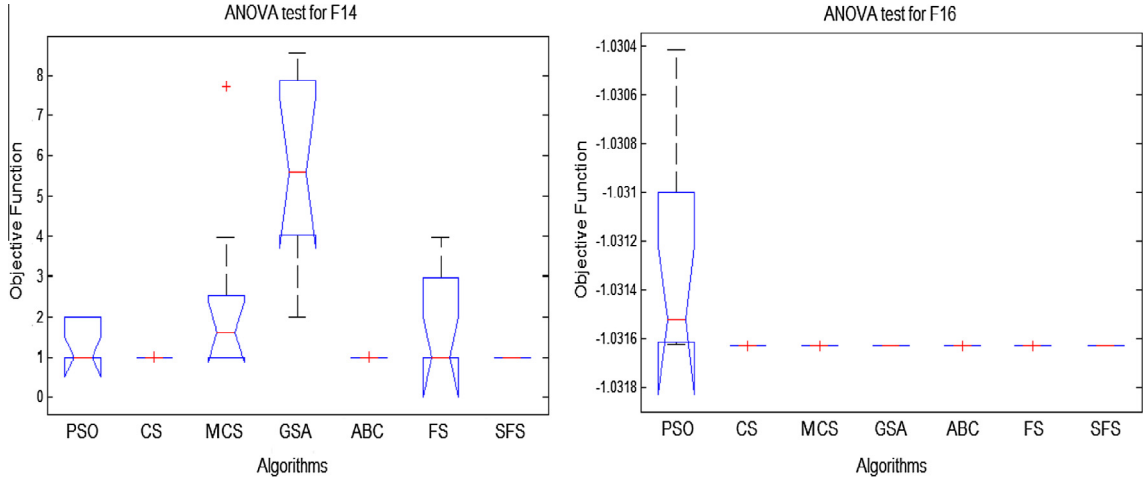


Fig. 5. ANOVA tests of the global minimum values, which are computed by using the PSO, CS, MCS, GSA, ABC, AMO and SFS for f14 and f16.

the only pioneer (see Fig. 6). The results of the second and third group can explicitly reflect the weakness of other well-known metaheuristic algorithms to overcome local minimums. The acquired results on the fourth group show that although all algorithms have poor performance on f_{15} benchmark function, comparing with other algorithms, SFS is superior in solving f_{14}, f_{16}, f_{17} and f_{18} benchmark functions. For the last group, however, SFS could find the global solutions for both functionsf_{19}andf_{20}. Since these functions are not quite hard optimization problems, some algorithms like DE, GSA and AMO could solve these functions with approximate solutions.

5.2.1. SFS search behavior and parameter analysis

In this section, the effect of SFS parameter on the approximation error, as well as the learning behavior of SFS are experimentally studied. To evaluate the SFS learning mechanism, three configurations which differ on activated learning mechanisms are considered. The first one corresponds to a coalition of agents without

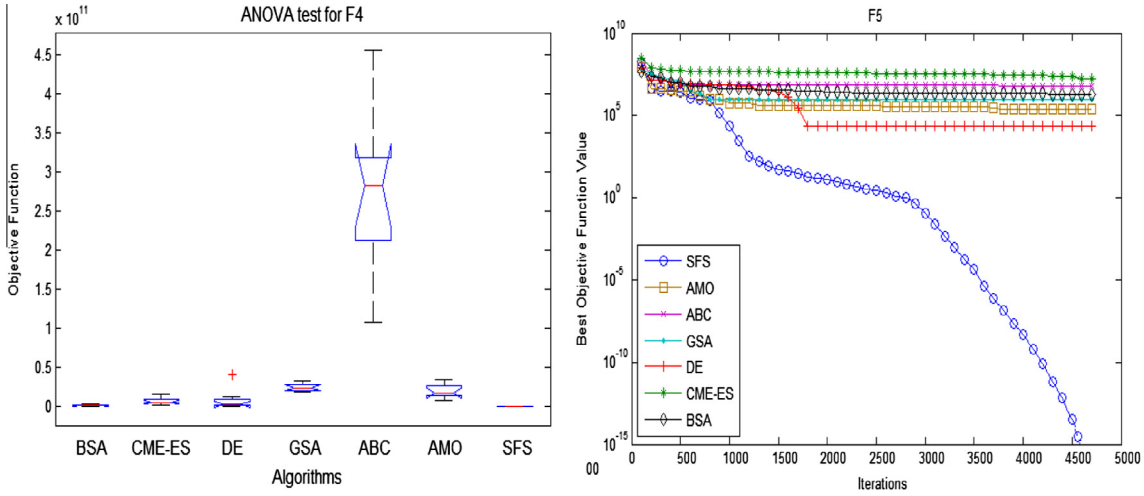


Fig. 6. ANOVA test and graphical results of functions F4 and F5 respectively.

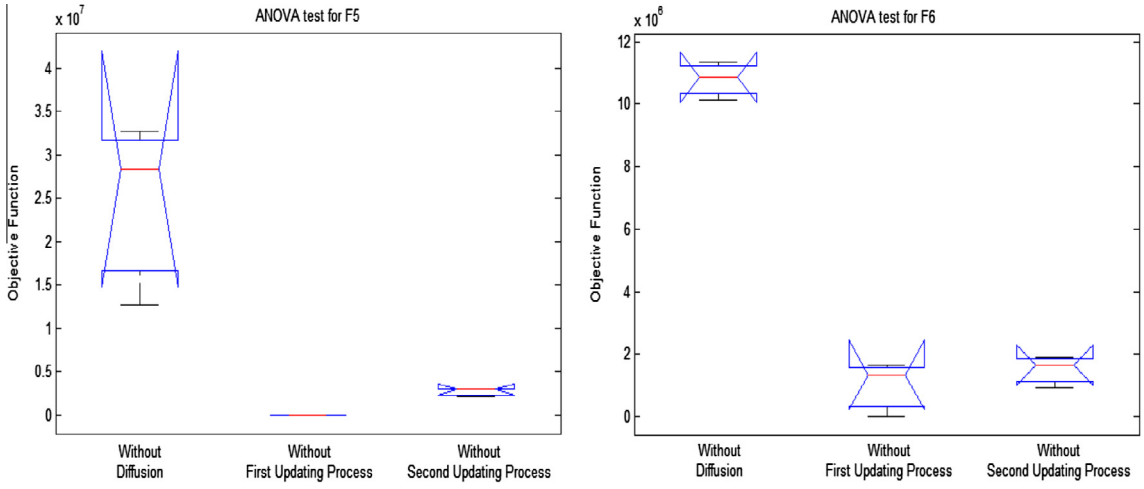


Fig. 7. ANOVA tests for SFS parts.

Table 8
The maximum iteration for each classic function.

Functions	Iterations	Functions	Iterations
f_{01}	1900	f_{11}	9000
f_{02}	2150	f_{12}	6000
f_{03}	3000	f_{13}	10,000
f_{04}	15,000	f_{14}	15,000
f_{05}	4700	f_{15}	4500
f_{06}	6300	f_{16}	15,000
f_{07}	5300	f_{17}	8500
f_{08}	7000	f_{18}	9000
f_{09}	15,500	f_{19}	10,200
f_{10}	5000	f_{20}	8000

Table 9
The control parameters of search algorithms.

Name	Specification	Population size
BSA	*mixrate* = 1.00 as recommended in [32]	100
CMA-ES	$\sigma = 0.25$, $\mu = \left\lfloor \frac{4+\lfloor 2\log(N)\rfloor}{2} \right\rfloor$ have been recommended in [33]	100
DE	$F = 0.5$ and $CR = 0.9$ in accordance given in [37]	100
GSA	$G_o = 100$ and $\alpha = 20$, and T is set to total iterations [9]	100
ABC	Limit = $50D$ has been used as recommended in [7]	100
AMO	The number of animals in each group was set to 5 [36]	100
SFS	Maximum diffusion level is set to 1	100

Diffusion learning mechanism. In the second configuration, the agents have the capacity to individually learn without first updating the process. In the last configuration, SFS has learnt without a second update in the process. These three configurations of SFS have been tested on classic and modern benchmark functions. The classic functions used in this experiment were f_{01}, f_{05} and f_{04}^{CEC}, f_{05}^{CEC}, f_{06}^{CEC} selected from CEC 2010 benchmark functions. The iteration number of each function was evaluated according to its previous condition. Table 11 reports the average results of 10 runs for each configuration.

The obtained results from Table 11 prove that each part in SFS can significantly effect the quality of final results, and it shows that all parts together make a coherent system to solve optimization problems (see Fig. 7).

As stated before, since FS has many parameters, we have introduced SFS which is just sensitive to a parameter called maximum diffusion number (MDN). Empirical results have implied that the diffusion number can influence the resulting performance by a varying amount, depending on the problem being optimized. Some test functions show considerably improved performance as MDN is

Table 10
Minimization result of benchmark functions $f01$–$f20$ for different algorithms

Functions	Algorithm	BSA	CMA-ES	DE	GSA	ABC	AMO	SFS
f_{01}	Mean	4.8469E−06	6.8632E+07	8.2863E−17	7.2092E+06	4.3641E−16	0	**0**
	StDev	3.3491E−06	3.2056E+07	9.7151E−17	1.4321E+06	9.0985E−17	0	**0**
	Rank	5	7	3	6	4	1	**1**
f_{02}	Mean	5.9635	223.5477	159.7944	6.6662	0	6.0036E−11	**0**
	StDev	2.4086	84.8192	24.5465	1.6281	0	1.8940E−10	**0**
	Rank	4	7	6	5	1	3	**1**
f_{03}	Mean	1.1438E−09	9.4655	5.3291E−15	1.3771E−09	3.0198E−14	3.5527E−15	**3.5527E−15**
	StDev	6.0197E−10	8.3909	1.8724E−15	1.6860E−10	3.4526E−15	0	**0**
	Rank	5	7	3	6	4	1	**1**
f_{04}	Mean	1.2156E+09	6.7160E+09	1.4413E+08	2.3939E+10	2.7032E+11	2.0024E+10	**4.3281E−20**
	StDev	8.7094E+08	4.5880E+09	3.2666E+08	4.7304E+09	9.9930E+10	8.6509E+09	**1.6711E−20**
	Rank	3	4	2	6	7	5	**1**
f_{05}	Mean	1.1127E+06	3.2207E+07	6.9659E+05	1.8744E+06	5.7088E+06	6.7998E+05	**0**
	StDev	1.0604E+06	1.1619E+07	6.7155E+05	8.4156E+05	1.8671E+06	5.7544E+05	**0**
	Rank	4	7	3	5	6	2	**1**
f_{06}	Mean	5.4374E+03	1.5869E+07	20.8572	4.1346E+05	3.3338E+06	1.9669E+05	**3.5531E−09**
	StDev	5.4528E+03	1.4141E+05	0.0663	5.4665E+05	6.7961E+05	3.0303E+05	**1.1235E−09**
	Rank	3	7	2	5	6	4	**1**
f_{07}	Mean	894.5796	1.3328E+06	1.1636E−22	4.1009E+04	6.5498E+07	0	**0**
	StDev	1.3173E+03	1.4065E+06	1.0730E−22	3.9536E+03	4.1649E+07	0	**0**
	Rank	4	6	3	5	7	1	**1**
f_{08}	Mean	1.2347E+05	2.1351E+09	4.5319E−21	1.8478E+07	1.6821E+06	1.3801E+06	**0**
	StDev	1.8343E+05	3.8577E+09	8.2374E−21	4.7711E+07	1.0682E+06	1.6550E+06	**0**
	Rank	3	7	2	6	5	4	**1**
f_{09}	Mean	4.5688E+03	3.7015E+04	2.0942E+03	4.9286E+04	2.9312E+05	5.0347E+04	**5.5108E−26**
	StDev	2.8090E+03	2.0906E+04	1.8450E+03	1.7239E+04	6.9067E+04	2.7133E+04	**1.6724E−26**
	Rank	3	4	2	5	7	6	**1**
f_{10}	Mean	27.7971	82.1360	166.1583	37.5427	29.2470	29.4090	**14.0978**
	StDev	3.0713	24.7454	9.9685	4.4786	4.1242	2.7395	**3.8297**
	Rank	2	6	7	5	3	4	**1**
f_{11}	Mean	1.9208	52.7430	20.6257	2.3254	4.8453	2.9187	**7.1054E−15**
	StDev	1.1409	0.1785	0.1159	0.9900	0.8090	0.4007	**0**
	Rank	2	7	6	3	5	4	**1**
f_{12}	Mean	0.0030	2.0843	4.9146E−29	4.3179E−18	14.5151	0	**0**
	StDev	0.0019	1.5671	3.9724E−29	8.7855E−19	6.8952	0	**0**
	Rank	5	6	3	4	7	1	**1**
f_{13}	Mean	0.2250	445.8593	1.1960	2.0048E+03	5.1361	2.6539	**0**
	StDev	0.3140	402.7247	1.9257	586.7841	3.5850	1.6702	**0**
	Rank	2	6	3	7	5	4	**1**
f_{14}	Mean	4.2395E+04	1.7432E+04	1.8530E+04	8.2062E+04	3.1586E+05	1.1510E+05	**8.8622E−25**
	StDev	1.5834E+04	1.1117E+04	7.9109E+03	4.0351E+04	1.0169E+05	4.7610E+04	**2.9666E−25**
	Rank	4	2	3	5	7	6	**1**
f_{15}	Mean	50.9949	42.7921	174.4380	23.5421	61.9689	44.9184	**21.7987**
	StDev	6.2798	4.9626	8.7652	4.1871	4.2834	5.3825	**3.8652**
	Rank	5	3	7	2	6	4	**1**
f_{16}	Mean	3.2806	55.7207	39.5581	0.0889	12.9872	8.7969	**1.3145E−14**
	StDev	2.1032	32.9190	20.7530	0.0128	3.1057	2.0553	**1.7161E−15**
	Rank	3	7	6	2	5	4	**1**
f_{17}	Mean	0.0188	0.0024	1.4357E−28	5.4468E−18	3.8360	1.5146E−29	**0**
	StDev	0.0128	0.0020	1.4056E−28	9.4311E−19	2.0564	2.4388E−29	**0**
	Rank	6	5	3	4	7	2	**1**
f_{18}	Mean	7.5503	3.6786E+03	0.3987	312.9730	8.1174	1.6596	**1.9722E−29**
	StDev	2.9290	5.7331E+03	1.2607	705.7324	3.5365	3.3689	**0**
	Rank	4	7	2	6	5	3	**1**
f_{19}	Mean	0.4749	0.0074	7.6480E−26	8.0626E−18	542.0824	1.1620E−27	**0**
	StDev	0.2948	0.0120	1.3953E−25	2.4198E−18	259.9031	5.4455E−28	**0**
	Rank	6	5	3	4	7	2	**1**
f_{20}	Mean	18.3278	24.4296	8.0986E−26	16.2356	0.1212	0.0024	**0**
	StDev	19.1371	1.9710	8.5162E−26	0.2309	0.2115	0.0036	**0**
	Rank	6	7	2	5	4	3	**1**
Average rank		3.9500	5.8500	3.5500	4.8000	5.4000	3.2000	**1**
Overall rank		4	7	3	5	6	2	**1**

Table 11
Investigation of SFS parts in final results.

Function	Without diffusion		Without first updating process		Without second updating process	
	Mean	Std	Mean	Std	Mean	Std
f_{01}	2.3832E+004	3.5797E+003	5.8276E−308	0	0	0
f_{05}	2.0946E+007	6.0629E+006	18.0299	5.8164	6.6682	1.8424
f_{04}^{CEC}	1.0647E+013	3.5095E+012	5.2037E−019	4.8530E−020	2.2312E+009	8.9182E+008
f_{05}^{CEC}	2.4651E+007	1.0507E+007	144.6729	27.0263	2.6899E+006	5.1088E+005
f_{06}^{CEC}	1.0779E+007	5.9898E+005	9.9555E+005	8.7562E+005	1.4925E+006	5.0222E+005

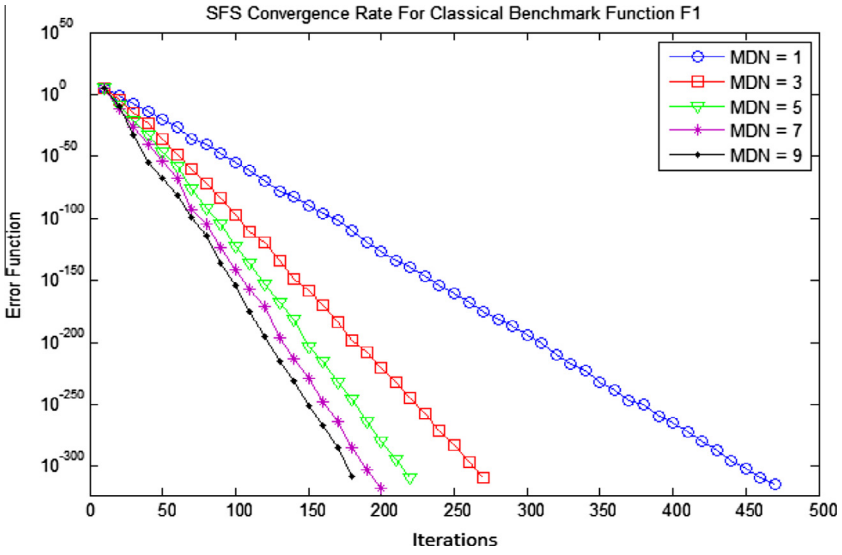


Fig. 8. Effect of MDN on convergence rate for classical benchmark function F1.

increased, while others tend to be better optimized only by a slight increase in MDN. However, our experimental results in previous sections showed that while SFS could usually find the global optimum point for MDN = 1, there seems to be a definitive value for MDN that is optimal across all problems. While it may certainly be beneficial to tune this parameter based on the problem at hand, generally speaking it is of relative importance. On the other hand, increasing MDN causes us to face a trade-off between time consumption and convergence rate. Figs. 8 and 9, shows the effect of MDN on the convergence rate and time consumption for f_{01} respectively.

5.3. Experiment III – Constrained engineering design optimization problem

So far, the performance of SFS algorithm is investigated on the type of problems called unconstrained problems. Other types of problems which are of the most important and ubiquitous types of engineering optimization problems are constrained global optimization problems. The general constrained global optimization problem with inequality, equality, lower bound, and upper bound constraints is defined as:

$$\begin{aligned} &\min f(x) \\ &\text{s.t } g_j(x) = 0, \quad j = 1,2,\ldots,p, \\ &\quad g_j(x) \leqslant 0, \quad j = p+1,\ldots,m, \\ &\quad l_i \leqslant x_i \leqslant u_i, \quad i = 1,2,\ldots,n. \end{aligned} \tag{19}$$

where $f(x)$ is the objective function, $g_j(x) = 0$ are the equality constraints, $g_j(x) \leqslant 0$ are inequality constraints, l_i and u_i are the lower bound and the upper bound of x_i, respectively [38].

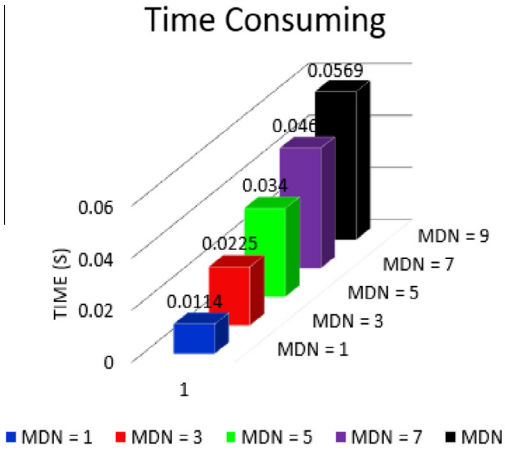


Fig. 9. Effect of MDN on time consumption for each cycle on classical benchmark function F1.

In order to study the performance of solving the real-world engineering design optimization problems, the proposed SFS algorithm is applied to three well-known constrained engineering design problems. In this work we employed the same constraint handling mechanism used in [39], and the results of 30 independent runs on SFS were placed at Tables 12–17.

5.3.1. Tension/compression spring design problem

The tension/compression spring design problem (as shown in Fig. 10) is described in Belegundu [40]. The design optimization

Table 12
Comparing of the best solution for tension/compression spring design problem found by different algorithms.

	MPM [40]	GA [41]	CEPSO [42]	CEDE [43]	WCA [44]	SFS
w	0.05000	0.051989	0.051728	0.051609	0.051689	0.051689060916152
d	0.31590	0.363965	0.357644	0.354714	0.356522	0.356717735791209
L	14.2500	10.890522	11.244543	11.410831	11.300410	11.288965986603480
g_1	−0.000014	−0.000013	−8.25E−04	−3.90E−05	−1.65E−13	−2.2204E−16
g_2	−0.003782	−0.000021	−2.52E−05	−1.83E−04	−7.90E−14	−1.1102E−16
g_3	−3.938302	−1.061338	−4.051306	−4.048627	−4.053399	−4.0538
g_4	−0.756067	−0.722698	−0.727085	−0.729118	−0.727864	−0.7277
f	0.0128334	0.0126810	0.012674	0.0126702	0.012665	**0.012665232788319**

Table 13
Statistical results of different approaches for tension/compression spring design problem.

	MPM [40]	GA [41]	CEPSO [42]	CEDE [43]	WCA [44]	SFS
Best	0.0128334	0.0126810	0.012674	0.0126702	0.012665	**0.012665232788319**
Mean	NA	0.0127420	0.012730	0.0126703	0.012746	**0.012665232788319**
Worst	NA	0.0129730	0.012924	0.0126790	0.012952	**0.012665232788319**
Std	NA	5.9E−05	1.58E−05	2.70E−05	8.06E−06	**1.5858E−16**
NFEs	NA	80,000	240,000	204,800	11,750	**100,000**

Table 14
Comparing of the best solution for pressure vessel design problem found by different algorithms.

	GAFR [41]	CEPSO [42]	CEDE [43]	PSO [46]	NIDP [45]	SFS
T_s	0.8125	0.8125	0.8125	0.8125	1.1250	0.8125 (13×0.0625)
T_h	0.4375	0.4375	0.4375	0.4375	0.6250	0.4375 (7×0.0625)
R	42.097398	42.0913	42.0984	42.0984	48.3807	42.098445595854920
L	176.65405	176.7465	176.7465	176.6366	11.7449	176.6365958424395
g_1	−0.000020	−1.37E−06	−6.67E−07	−8.8000E−07	−0.1913	0
g_2	−0.035891	−3.59E−04	−3.58E−02	−0.0359	−0.1634	−0.0359
g_3	−27.886075	−118.7687	−3.705123	3.1227	−75.875	−4.6566E−10
g_4	−63.345953	−63.2535	−63.3623	−63.3634	−128.2551	−63.3634
f	6059.9463	6061.0777	6059.7340	6059.7143	8048.6190	**6059.714335048436**

Table 15
Statistical results of different approaches for pressure vessel design problem.

	GAFR [41]	CEPSO [42]	CEDE [43]	PSO [46]	NIDP [45]	SFS
Best	6059.9463	6061.0777	6059.7340	6059.7143	8048.6190	**6059.714335048436**
Mean	6177.2533	6147.1332	6085.2303	6066.0311	NA	**6059.714335048436**
Worst	6469.3220	6363.8041	6371.0455	NA	NA	**6059.714335048436**
Std	130.9297	86.4500	43.0130	12.2718	NA	**9.5869E−13**
NFEs	80,000	240,000	27,500	60,000	NA	**50,000**

Table 16
Comparing of the best solution for Welded Beam design problem found by different algorithms.

	Deb [49]	Coello [50]	Montes [41]	He [51]	Kaveh [52]	SFS
w	0.248900	0.208800	0.205986	0.202369	0.205700	0.205729639786079
L	6.173000	3.420500	3.471328	3.544214	3.471131	3.470488665628001
d	8.178900	8.997500	9.020224	9.048210	9.036683	9.036623910357635
h	0.253300	0.210000	0.206480	0.205723	0.205731	0.205729639786079
g_1	−5.7586E+03	−0.3378	−0.1030	−13.6555	−0.0846	−1.8190E−012
g_2	−255.5769	−353.9026	−0.2317	−75.8141	−0.5907	−1.4552E−011
g_3	−0.0044	−0.0012	−4.940E−004	−0.0034	−3.1000E−05	−2.7756E−017
g_4	−2.9829	−3.4119	−3.4300	−3.4246	−3.4329	−3.4330
g_5	−0.1239	−0.0838	−0.0810	−0.0774	−0.0807	−0.0807
g_6	−0.2342	−0.2356	−0.2355	−0.2356	−0.2355	−0.2355
g_7	−4.4653E+03	−363.2324	−58.6469	−4.4729	−0.1448	−1.8190e−012
f	2.433116	1.748309	1.728226	1.728024	1.724918	**1.724852308597364**

Table 17
Statistical results of different approaches for Welded Beam design problem.

	Deb [49]	Coello [50]	Montes [41]	He [51]	Kaveh [52]	SFS
Best	2.433116	1.748309	1.728226	1.728024	1.724918	**1.724852308597364**
Mean	NA	1.771973	1.792654	1.748831	1.729752	**1.724852308597364**
Worst	NA	1.785835	1.993408	1.782143	1.775961	**1.724852308597364**
Std	NA	0.011220	0.074713	0.012926	0.009200	**7.7087E−016**
NFEs	NA	900,000	80,000	30,000	17,600	**24,000**

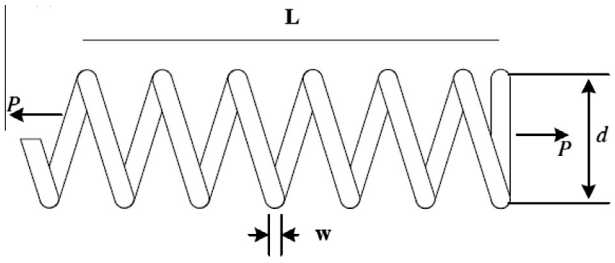


Fig. 10. Tension/compression spring design problem.

problem involves three continuous variables and four nonlinear inequality constraints.

The mathematical model of the spring design problem is expressed as:

$$\min f(w,d,L) = (L+2)w^2 d \tag{20}$$

Subject to

$$\begin{aligned} g_1 &= 1 - \frac{d^3 L}{71785 w^4} \leqslant 0 \\ g_2 &= \frac{d(4d-w)}{12566 w^3 (d-w)} + \frac{1}{5108 w^2} - 1 \leqslant 0 \\ g_3 &= 1 - \frac{140.45 w}{d^2 L} \leqslant 0 \\ g_4 &= \frac{2(w+d)}{3} - 1 \leqslant 0 \end{aligned} \tag{21}$$

where w, d and L are recognized as wire diameter, the mean coil diameter, and the length (or number of coils) respectively. The following limits for these parameters are:

$$0.05 \leqslant w \leqslant 2.0, \quad 0.25 \leqslant d \leqslant 1.3, \quad 2.0 \leqslant L \leqslant 15.0 \tag{22}$$

This problem has already been solved by several researchers, including Belegundu [40], who used Mathematical Programming Methods (MPM), Coello and Mezura-Montes [41], who used genetic algorithm (GA), Krohling and Coelho [42], who employed an approach of Coevolutionary Particle Swarm Optimization (CEPSO), Huang et al. [43] proposed an effective Coevolutionary Differential Evolution (CEDE), and Eskandar et al. [44], who applied a Water Cycle Algorithm (WCA).

The best solutions obtained by SFS in this work for the spring design problem were compared with the five best solutions reported in the literature, and are presented in Table 12. Their statistical results are listed in Table 13.

As shown in Tables 12 and 13, the searching quality of the proposed SFS is superior to that of the other five methods. Moreover, the standard deviation of the results by SFS is much smaller than that of other five stochastic approaches.

5.3.2. Pressure vessel design problem

The pressure vessel design problem (see Fig. 11) is described in Sandgren [45] who first proposed this problem. The design problem involves two discrete values and two continuous variables

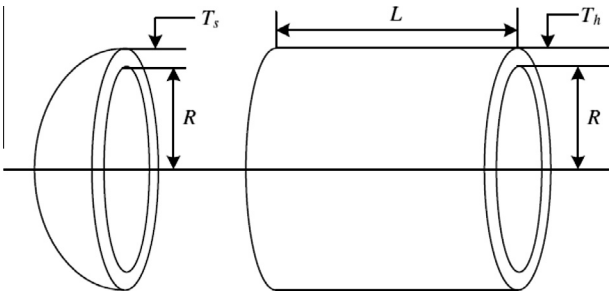


Fig. 11. Pressure vessel.

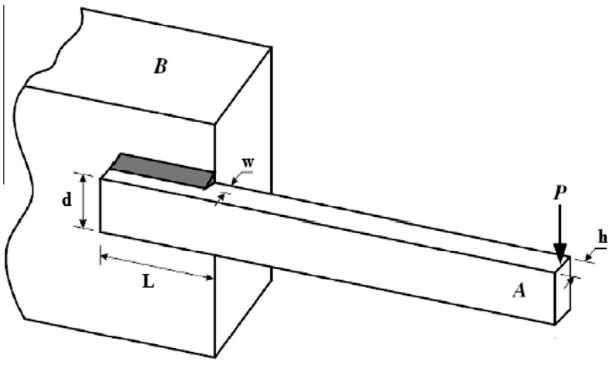


Fig. 12. Welded beam.

and four inequality constraints. The problem can be stated as follows: (see Fig. 12)

$$\begin{aligned} \min f(T_s,T_h,R,L) = {} & 0.6224 T_s R L + 1.7781 T_h R^2 + 3.1661 T_s^2 L \\ & + 19.84 T_s^2 R \end{aligned} \tag{23}$$

Subject to

$$\begin{aligned} g_1 &= -T_s + 0.0193R \leqslant 0 \\ g_2 &= -T_h + 0.0095R \leqslant 0 \\ g_3 &= -\pi R^2 L - \frac{4}{3}\pi R^3 + 1{,}296{,}000 \leqslant 0 \\ g_4 &= L - 240 \leqslant 0 \end{aligned} \tag{24}$$

where T_s, T_h, R and L are the thickness, thickness of the head, the inner radius and the length of the cylindrical section of the vessel respectively. The restricted area is as follows:

$$1 \times 0.0625 \leqslant T_s, \quad T_h \leqslant 99 \times 0.0625, \quad \text{and} \quad 10 \leqslant R, L \leqslant 200.$$

Coello and Mezura-Montes [41] used a GA-based method to solve the pressure vessel design problem, Krohling and Coelho [42] applied Coevolutionary PSO to solve this problem, The CEDE algorithm is proposed by Huang et al. [43]. Also to solve this problem, Gao and Hailu [46] used a PSO based algorithm. Table 14 lists

the optimal solutions that have been determined by GA [41], CEPSO [42], CEDE [43], PSO [46], Nonlinear integer and discrete programming (NIDP) [45], as well as the proposed SFS in this study, and their statistical results are given in Table 15.

From Tables 14 and 15, with respect to GA, CEPSO, CEDE, and NIDP, SFS provided better results for pressure vessel design problem. Although the objective value derived by PSO algorithm is acceptable comparing with SFS, the reported value is not feasible. This is because the third constrained condition violated in the results of PSO.

5.3.3. Welded beam design problem

One of the standard constrained engineering problems is Welded beam (see Fig. 12) design optimization [47,48]. The problem involves four design variables including the width w and length L of the welded area, the depth h and thickness d of the main beam. The objective is to minimize the overall fabrication cost, under the appropriate constraints of shear stress τ, bending stress σ, buckling load P and maximum end deflection δ.

The problem can be written as:

$$\min f(w,L,d,h) = 1.10471w^2L + 0.04811dh(14.0+L) \tag{25}$$

Subject to:

$$\begin{aligned} g_1 &= w - h \leqslant 0 \\ g_2 &= \delta - 0.25 \leqslant 0 \\ g_3 &= \tau - 13,600 \leqslant 0 \\ g_4 &= \sigma - 30,000 \leqslant 0 \\ g_5 &= 0.10471w^2 + 0.04811hd(14+L) - 5.0 \leqslant 0 \\ g_6 &= 0.125 - w \leqslant 0 \\ g_7 &= 6000 - P \leqslant 0 \end{aligned} \tag{26}$$

where

$$\begin{aligned} &\sigma = \frac{504,000}{hd^2} \quad Q = 6000\left(14+\frac{L}{2}\right) \\ &D = \frac{1}{2}\sqrt{L^2+(w+d)^2} \quad J = \sqrt{2}wL\left[\frac{L^2}{6}+\frac{(w+d)^2}{2}\right] \\ &\delta = \frac{65,856}{30,000hd^3} \quad \beta = \frac{QD}{J} \\ &\alpha = \frac{6000}{\sqrt{2}wL} \quad \tau = \sqrt{\alpha^2+\frac{\alpha\beta L}{D}+\beta^2} \\ &P = 0.61423\times 10^6\frac{dh^3}{6}\left(1-\frac{d\sqrt{30/48}}{28}\right) \end{aligned} \tag{27}$$

The simple limits or bounds are $0.1 \leqslant L, d \leqslant 10$ and $0.1 \leqslant w, h \leqslant 2.0$.

The welded beam structure is a practical design problem that has been often used as a benchmark problem for testing different optimization methods. Deb [49], Coello [50] and Coello and Mezura-Montes [41] solved this problem using GA-based methods. Radgsdell and Phillips [48] compared the optimal results of different optimization methods that were mainly based on mathematical optimization algorithms. These methods, are APPROX (Griffith and Stewart's successive linear approximation), DAVID (Davidon–Fletcher–Powell with a penalty function), SIMPLEX (Simplex method with a penalty function), and RANDOM (Richardson's random method) algorithms. Also, This problem was solved by He and Wang [51] using CPSO, Montes and Coello [53] employing evolution strategies and the Kaveh and Talatahari [52] using the ACO. The comparison of results, are shown in Table 16. The statistical simulation results are summarized in Table 17. From Table 17, it can be seen that, the standard deviation of the results by SFS in 30 independent runs is very small.

6. Conclusions

In this paper, we propose a new model based on random fractals to solve both constrained and unconstrained global optimization problems with continuous variables. Our method employs two main processes including: Diffusion process and Update process. During the first process, each agent in the group generates some new agents around its current position, and forms the diffusion process. The second process simulates the updating position of each agent based on some statistical approaches. To verify the performance of our algorithm, some benchmark functions including spectrum classic, modern and engineering design functions which are commonly used in literature, have been used. Surprisingly, obtained results show that our new algorithm clearly outperforms some well-known algorithms from literature. The proposed system can contribute to answering many fundamental and practical problems in many fields such as machine learning, bioinformatics, image processing and generally any field in need of optimization. Improving the classification performance, being considered as a powerful feature selection, determining RNA secondary predictions, finding high order SNP interactions, and improving image segmentation are some possible applications contemplated as future studies. However, further research is required to examine the efficiencies of the proposed algorithm on other real world and large scale optimization problems.

Acknowledgments

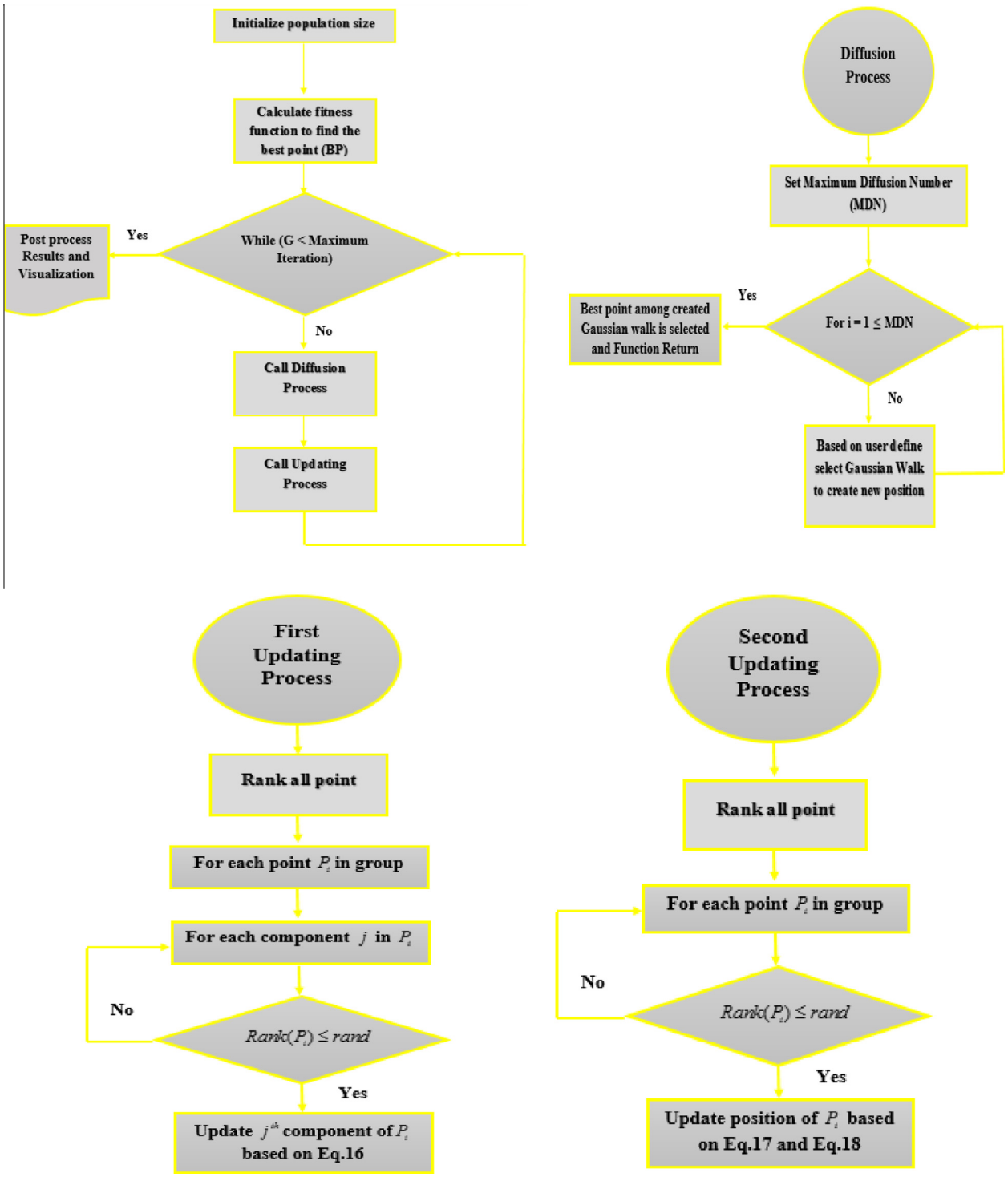
The author would like to thank Prof. F. Glover from University of Colorado Boulder, and the anonymous reviewers for their constructive comments and suggestions.

Appendix A

All codes used in this paper are downloaded from below addresses:

PSO: http://www.mathworks.com/matlabcentral/fileexchange/7506-particle-swarm-optimization-toolbox
CS: http://www.mathworks.com/matlabcentral/fileexchange/29809-cuckoo-search-cs-algorithm
MCS: https://code.google.com/p/modified-cs/
DE: http://academic.csuohio.edu/simond/bbo/
GSA: http://www.mathworks.com/matlabcentral/fileexchange/27756-gravitational-search-algorithm-gsa
ABC: http://mf.erciyes.edu.tr/abc/
BSA: http://www.mathworks.com/matlabcentral/fileexchange/44842-backtracking-search-optimization-algorithm
CMA-ES: https://www.lri.fr/~hansen/cmaes_inmatlab.html
AMO: You can obtain this code through requesting from Xiangtao Li: Email: lixt314@nenu.edu.cn
SFS: http://www.mathworks.com/matlabcentral/fileexchange/47565-stochastic-fractal-search–sfs-

Appendix B



References

[1] E.-G. Talbi, Metaheuristics: From Design to Implementation, vol. 74, John Wiley & Sons, 2009.
[2] X.-S. Yang, S. Deb, Engineering optimisation by cuckoo search, Int. J. Math. Model. Numer. Optim. 1 (4) (2010) 330–343.
[3] A. Lazar, Heuristic knowledge discovery for archaeological data using genetic algorithms and rough sets1, Heuristics Optim. Knowl. Discov. (2002) 263.
[4] S.J. Russell et al., Artificial Intelligence: A Modern Approach, vol. 74, Prentice hall Englewood Cliffs, 1995.
[5] M. Mitchell, An Introduction to Genetic Algorithms (Complex Adaptive Systems), 1998.
[6] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995, MHS'95, IEEE, 1995.

[7] D. Karaboga, An Idea Based on Honey Bee Swarm for Numerical Optimization, Techn. Rep. TR06, Erciyes Univ. Press, Erciyes, 2005.
[8] M. Dorigo, G. Di Caro, Ant colony optimization: a new meta-heuristic, in: Proceedings of the 1999 Congress on Evolutionary Computation, 1999, CEC 99, IEEE, 1999.
[9] E. Rashedi, H. Nezamabadi-Pour, S. Saryazdi, GSA: a gravitational search algorithm, Inf. Sci. 179 (13) (2009) 2232–2248.
[10] H. Salimi, D. Giveki, Farsi/Arabic handwritten digit recognition based on ensemble of SVD classifiers and reliable multi-phase PSO combination rule, Int. J. Doc. Anal. Recogn. (IJDAR) (2012) 1–16.
[11] D. Giveki et al., Detection of erythemato-squamous diseases using AR-CatfishBPSO-KSVM, Signal Image Process. 2 (4) (2012) 57–72.
[12] H. Salimi et al., Extended Mixture of MLP Experts by Hybrid of Conjugate Gradient Method and Modified Cuckoo Search, 2012. arXiv Preprint arXiv:1202.3887.
[13] M.J. Abdi, H. Salimi, Farsi handwriting recognition with mixture of RBF experts based on particle swarm optimization, Int. J. Inf. Sci. Comput. Math. 2 (2) (2010) 129–136.
[14] D. Giveki et al., Automatic Detection of Diabetes Diagnosis using Feature Weighted Support Vector Machines based on Mutual Information and Modified Cuckoo Search, 2012. arXiv preprint arXiv:1201.2173.
[15] M. Aflakparast et al., Cuckoo search epistasis: a new method for exploring significant genetic interactions, Heredity (2014).
[16] T. Liao et al., Ant colony optimization for mixed-variable optimization problems, IEEE. Trans. Evol. Comput. 18 (4) (2014) 503–518.
[17] M. Lozano et al., A hybrid metaheuristic for the cyclic antibandwidth problem, Knowl.-Based Syst. 54 (2013) 103–113.
[18] B.B. Mandelbrot, The Fractal Geometry of Nature, Times Books, 1983.
[19] T. Vičzek, Fractal Growth Phenomena, World Scientific Publishing Company Incorporated, 1992.
[20] M.F. Barnsley, S. Demko, Iterated function systems and the global construction of fractals, Proc. Roy. Soc. Lond. A. Math. Phys. Sci. 399 (1817) (1985) 243–275.
[21] P. Grassberger, I. Procaccia, Characterization of strange attractors, Phys. Rev. Lett. 50 (5) (1983) 346–349.
[22] P. Prusinkiewicz, Graphical applications of L-systems, in: Proceedings of Graphics Interface, 1986.
[23] J. Cannon, W. Floyd, W. Parry, Finite subdivision rules, Conformal Geom. Dynam. Am. Math. Soc. 5 (8) (2001) 153–196.
[24] K.J. Falconer, Random fractals, Math. Proc. Cambridge Philos. Soc 100 (3) (1986) 559–582.
[25] T.t. Witten, L. Sander, Diffusion-limited aggregation, Phys. Rev. B 27 (9) (1983) 5686.
[26] L. Niemeyer, L. Pietronero, H. Wiesmann, Fractal dimension of dielectric breakdown, Phys. Rev. Lett. 52 (12) (1984) 1033–1036.
[27] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Trans. Evol. Comput. 3 (2) (1999) 82–102.
[28] S. Walton et al., Modified cuckoo search: a new gradient free optimisation algorithm, Chaos, Solitons Fract. 44 (9) (2011) 710–718.
[29] M. Clerc, J. Kennedy, The particle swarm-explosion, stability, and convergence in a multidimensional complex space, IEEE Trans. Evol. Comput. 6 (1) (2002) 58–73.
[30] X.-S. Yang, S. Deb, Cuckoo search via Lévy flights, in: World Congress on Nature & Biologically Inspired Computing, 2009, NaBIC 2009, IEEE, 2009.
[31] K. Tang et al., Benchmark Functions for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization, Nature Inspired Computation and Applications Laboratory, USTC, China, 2009.
[32] P. Civicioglu, Backtracking search optimization algorithm for numerical optimization problems, Appl. Math. Comput. 219 (15) (2013) 8121–8144.
[33] N. Hansen, S.D. Müller, P. Koumoutsakos, Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES), Evol. Comput. 11 (1) (2003) 1–18.
[34] P. Civicioglu, Transforming geocentric cartesian coordinates to geodetic coordinates by using differential search algorithm, Comput. Geosci. 46 (2012) 229–247.
[35] B. Akay, D. Karaboga, A modified artificial bee colony algorithm for real-parameter optimization, Inf. Sci. 192 (2012) 120–142.
[36] X. Li, J. Zhang, M. Yin, Animal migration optimization: an optimization algorithm inspired by animal migration behavior, Neural Comput. Appl. 24 (7–8) (2014) 1867–1877.
[37] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (4) (1997) 341–359.
[38] W. Long et al., An effective hybrid cuckoo search algorithm for constrained global optimization, Neural Comput. Appl. 25 (4) (2014) 911–926.
[39] A.H. Gandomi et al., Bat algorithm for constrained optimization tasks, Neural Comput. Appl. 22 (6) (2013) 1239–1255.
[40] A.D. Belegundu, J.S. Arora, A study of mathematical programming methods for structural optimization, Int. J. Numer. Methods Eng. I: Theory 21 (9) (1985) 1583–1599.
[41] C.A. Coello Coello, E. Mezura Montes, Constraint-handling in genetic algorithms through the use of dominance-based tournament selection, Adv. Eng. Inform. 16 (3) (2002) 193–203.
[42] R.A. Krohling, L. dos Santos Coelho, Coevolutionary particle swarm optimization using Gaussian distribution for solving constrained optimization problems, IEEE Trans. Syst. Man Cybern. Part B: Cybern. 36 (6) (2006) 1407–1416.
[43] F.-z. Huang, L. Wang, Q. He, An effective co-evolutionary differential evolution for constrained optimization, Appl. Math. Comput. 186 (1) (2007) 340–356.
[44] H. Eskandar et al., Water cycle algorithm–A novel metaheuristic optimization method for solving constrained engineering optimization problems, Comput. Struct. (2012).
[45] E. Sandgren, Nonlinear integer and discrete programming in mechanical design optimization, J. Mech. Des. 112 (1990) 223.
[46] L. Gao, A. Hailu, Comprehensive learning particle swarm optimizer for constrained mixed-variable optimization problems, Int. J. Comput. Intell. Syst. 3 (6) (2010) 832–842.
[47] L.C. Cagnina, S.C. Esquivel, C.A.C. Coello, Solving engineering optimization problems with the simple constrained particle swarm optimizer, Informatica (Slovenia) 32 (3) (2008) 319–326.
[48] K. Ragsdell, D. Phillips, Optimal design of a class of welded structures using geometric programming, J. Eng. Ind. 98 (1976) 1021.
[49] K. Deb, Optimal design of a welded beam via genetic algorithms, AIAA J. 29 (11) (1991) 2013–2015.
[50] C.A. Coello Coello, Use of a self-adaptive penalty approach for engineering optimization problems, Comput. Ind. 41 (2) (2000) 113–127.
[51] Q. He, L. Wang, An effective co-evolutionary particle swarm optimization for constrained engineering design problems, Eng. Appl. Artif. Intell. 20 (1) (2007) 89–99.
[52] A. Kaveh, S. Talatahari, An improved ant colony optimization for constrained engineering design problems, Eng. Comput. 27 (1) (2010) 155–182.
[53] E. Mezura-Montes, C.A.C. Coello, An empirical study about the usefulness of evolution strategies to solve constrained optimization problems, Int. J. Gen. Syst. 37 (4) (2008) 443–473.