GSA: a Gravitational Search Algorithm

Article *in* Information Sciences · June 2009

DOI: 10.1016/j.ins.2009.03.004 · Source: DBLP

3 authors:

Esmat Rashedi
Kerman Graduate University of Technology
40 PUBLICATIONS **1,826** CITATIONS
SEE PROFILE

Hossein Nezamabadi-pour
Shahid Bahonar University of Kerman
240 PUBLICATIONS **3,138** CITATIONS
SEE PROFILE

Saeid Saryazdi
Shahid Bahonar University of Kerman
35 PUBLICATIONS **1,791** CITATIONS
SEE PROFILE

Information Sciences 179 (2009) 2232–2248

Contents lists available at ScienceDirect

Information Sciences

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

GSA: A Gravitational Search Algorithm

Esmat Rashedi, Hossein Nezamabadi-pour *, Saeid Saryazdi

Department of Electrical Engineering, Shahid Bahonar University of Kerman, P.O. Box 76169-133, Kerman, Iran

ARTICLE INFO

Article history:
Received 21 April 2008
Received in revised form 11 January 2009
Accepted 8 March 2009

Keywords:
Optimization
Heuristic search algorithms
Gravitational Search Algorithm
Law of gravity

ABSTRACT

In recent years, various heuristic optimization methods have been developed. Many of these methods are inspired by swarm behaviors in nature. In this paper, a new optimization algorithm based on the law of gravity and mass interactions is introduced. In the proposed algorithm, the searcher agents are a collection of masses which interact with each other based on the Newtonian gravity and the laws of motion. The proposed method has been compared with some well-known heuristic search methods. The obtained results confirm the high performance of the proposed method in solving various nonlinear functions.

1. Introduction

In solving optimization problems with a high-dimensional search space, the classical optimization algorithms do not provide a suitable solution because the search space increases exponentially with the problem size, therefore solving these problems using exact techniques (such as exhaustive search) is not practical.

Over the last decades, there has been a growing interest in algorithms inspired by the behaviors of natural phenomena [5,8,17,19,21,32]. It is shown by many researchers that these algorithms are well suited to solve complex computational problems such as optimization of objective functions [6,36], pattern recognition [24,31], control objectives [2,16,20], image processing [4,27], filter modeling [15,23], etc. Various heuristic approaches have been adopted by researches so far, for example Genetic Algorithm [32], Simulated Annealing [21], Ant Colony Search Algorithm [5], Particle Swarm Optimization [17], etc. These algorithms are progressively analyzed or powered by researchers in many different areas [1,3,7,12,25,34]. These algorithms solve different optimization problems. However, there is no specific algorithm to achieve the best solution for all optimization problems. Some algorithms give a better solution for some particular problems than others. Hence, searching for new heuristic optimization algorithms is an open problem [35].

In this paper, a new optimization algorithm based on the law of gravity, namely Gravitational Search Algorithm (GSA) is proposed [28]. This algorithm is based on the Newtonian gravity: "Every particle in the universe attracts every other particle with a force that is directly proportional to the product of their masses and inversely proportional to the square of the distance between them".

This paper is organized as follows. Section 2 provides a brief review of heuristic algorithms. In Section 3, we introduce the basic aspects of our algorithm. In Section 4, the Gravitational Search Algorithm (GSA) and its characteristics are described. A comparative study is presented in Section 5 and, finally in Section 6 the experimental results are demonstrated.

* Corresponding author. Tel./fax: +98 341 3235900.
E-mail address: nezam@mail.uk.ac.ir (H. Nezamabadi-pour).

0020-0255/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2009.03.004

2. A review of heuristic algorithms

The word "heuristic" is Greek and means "to know", "to find", "to discover" or "to guide an investigation" [22]. Specifically, "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." (Russell and Norvig, 1995; [29])

Heuristic algorithms mimic physical or biological processes. Some of most famous of these algorithms are Genetic Algorithm, Simulated Annealing [21], Artificial Immune System [8], Ant Colony Optimization, Particle Swarm Optimization [17] and Bacterial Foraging Algorithm [11]. Genetic Algorithm, GA, are inspired from Darwinian evolutionary theory [32], Simulated Annealing, SA, is designed by use of thermodynamic effects [21], Artificial Immune Systems, AIS, simulate biological immune systems [8], Ant Colony Optimization, ACO, mimics the behavior of ants foraging for food [5], Bacterial Foraging Algorithm, BFA, comes from search and optimal foraging of bacteria [11,19] and Particle Swarm Optimization, PSO, simulates the behavior of flock of birds [3,17].

All of the above-mentioned heuristic algorithms have a stochastic behavior. However, Formato [9,10] has proposed a deterministic heuristic search algorithm based on the metaphor of gravitational kinematics that is called Central Force Optimization, CFO.

In some of stochastic algorithms, like SA, the search starts from a single point and continues in a sequential manner, however, most of the heuristic algorithms do search in a parallel fashion with multiple initial points, e.g. swarm based algorithms use a collection of agents similar to a natural flock of birds or fishes.

In a swarm based algorithm, each member executes a series of particular operations and shares its information with others. These operations are almost very simple, however their collective effect, known as swarm intelligence [5,33], produce a surprising result. Local interactions between agents provide a global result which permit the system to solve the problem without using any central controller. In this case, member operations including randomized search, positive feedback, negative feedback and multiple interactions, conduct to a self-organization situation [5].

One can recognize two common aspects in the population-based heuristic algorithms: exploration and exploitation. The exploration is the ability of expanding search space, where the exploitation is the ability of finding the optima around a good solution. In premier iterations, a heuristic search algorithm explores the search space to find new solutions. To avoid trapping in a local optimum, the algorithm must use the exploration in the first few iterations. Hence, the exploration is an important issue in a population-based heuristic algorithm. By lapse of iterations, exploration fades out and exploitation fades in, so the algorithm tunes itself in semi-optimal points. To have a high performance search, an essential key is a suitable tradeoff between exploration and exploitation. However, all the population-based heuristic algorithms employ the exploration and exploitation aspects but they use different approaches and operators. In other words, all search algorithms have a common framework.

From a different point of view, the members of a population-based search algorithm pass three steps in each iteration to realize the concepts of exploration and exploitation: self-adaptation, cooperation and competition. In the self-adaptation step, each member (agent) improves its performance. In the cooperation step, members collaborate with each other by information transferring. Finally, in the competition step, members compete to survive. These steps have usually stochastic forms, and could be realized in different ways. These steps, inspired from nature, are the principle ideas of the population-based heuristic algorithms. These concepts guide an algorithm to find a global optimum.

However, all population-based search algorithms provide satisfactory results but there is no heuristic algorithm that could provide a superior performance than others in solving all optimizing problems. In other words, an algorithm may solve some problems better and some problems worse than others [35]. Hence, proposing new high performance heuristic algorithms are welcome.

In this paper, our aim is to establish a new population-based search algorithm considering the mentioned aspects and based on the gravity rules. In the next section, a brief overview of gravitational force is given to provide a proper background and followed by explanation of GSA.

3. The law of gravity

The gravitation is the tendency of masses to accelerate toward each other. It is one of the four fundamental interactions in nature [30] (the others are: the electromagnetic force, the weak nuclear force, and the strong nuclear force). Every particle in the universe attracts every other particle. Gravity is everywhere. The inescapability of gravity makes it different from all other natural forces.

The way Newton's gravitational force behaves is called "action at a distance". This means gravity acts between separated particles without any intermediary and without any delay. In the Newton law of gravity, each particle attracts every other particle with a 'gravitational force' [14,30]. The gravitational force between two particles is directly proportional to the product of their masses and inversely proportional to the square of the distance between them [14]:

$$F = G\frac{M_1 M_2}{R^2}, \tag{1}$$

where F is the magnitude of the gravitational force, G is gravitational constant, M_1 and M_2 are the mass of the first and second particles respectively, and R is the distance between the two particles. Newton's second law says that when a force, F, is applied to a particle, its acceleration, a, depends only on the force and its mass, M [14]:

$$a = \frac{F}{M}. \tag{2}$$

Based on (1) and (2), there is an attracting gravity force among all particles of the universe where the effect of bigger and the closer particle is higher. An increase in the distance between two particles means decreasing the gravity force between them as it is illustrated in Fig. 1. In this figure, F_{1j} is the force that acting on M_1 from M_j and F_1 is the overall force that acts on M_1 and causes the acceleration vector a_1.

In addition, due to the effect of decreasing gravity, the actual value of the "gravitational constant" depends on the actual age of the universe. Eq. (3) gives the decrease of the gravitational constant, G, with the age [26]:

$$G(t) = G(t_0) \times \left(\frac{t_0}{t}\right)^{\beta}, \quad \beta < 1, \tag{3}$$

where $G(t)$ is the value of the gravitational constant at time t. $G(t_0)$ is the value of the gravitational constant at the first cosmic quantum-interval of time t_0 [26]. Three kinds of masses are defined in theoretical physics:

Active gravitational mass, M_a, is a measure of the strength of the gravitational field due to a particular object. Gravitational field of an object with small active gravitational mass is weaker than the object with more active gravitational mass.
Passive gravitational mass, M_p, is a measure of the strength of an object's interaction with the gravitational field. Within the same gravitational field, an object with a smaller passive gravitational mass experiences a smaller force than an object with a larger passive gravitational mass.
Inertial mass, M_i, is a measure of an object resistance to changing its state of motion when a force is applied. An object with large inertial mass changes its motion more slowly, and an object with small inertial mass changes it rapidly.

Now, considering the above-mentioned aspects, we rewrite Newton's laws.

The gravitational force, F_{ij}, that acts on mass i by mass j, is proportional to the product of the active gravitational of mass j and passive gravitational of mass i, and inversely proportional to the square distance between them. a_i is proportional to F_{ij} and inversely proportional to inertia mass of i. More precisely, one can rewrite Eqs. (1) and (2) as follows:

$$F_{ij} = G\frac{M_{aj} \times M_{pi}}{R^2}, \tag{4}$$

$$a_i = \frac{F_{ij}}{M_{ii}}, \tag{5}$$

where M_{aj} and M_{pi} represent the active gravitational mass of particle i and passive gravitational mass of particle j, respectively, and M_{ii} represents the inertia mass of particle i.

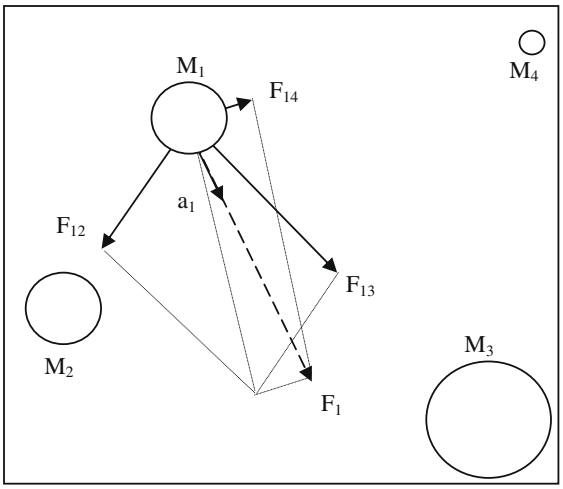

Fig. 1. Every mass accelerate toward the result force that act it from the other masses.

Although inertial mass, passive gravitational mass, and active gravitational mass are conceptually distinct, no experiment has ever unambiguously demonstrated any difference between them. The theory of general relativity rests on the assumption that inertial and passive gravitational mass are equivalent. This is known as the weak equivalence principle [18]. Standard general relativity also assumes the equivalence of inertial mass and active gravitational mass; this equivalence is sometimes called the strong equivalent principle [18].

4. Gravitational Search Algorithm (GSA)

In this section, we introduce our optimization algorithm based on the law of gravity [28]. In the proposed algorithm, agents are considered as objects and their performance is measured by their masses. All these objects attract each other by the gravity force, and this force causes a global movement of all objects towards the objects with heavier masses. Hence, masses cooperate using a direct form of communication, through gravitational force. The heavy masses – which correspond to good solutions – move more slowly than lighter ones, this guarantees the exploitation step of the algorithm.

In GSA, each mass (agent) has four specifications: position, inertial mass, active gravitational mass, and passive gravitational mass. The position of the mass corresponds to a solution of the problem, and its gravitational and inertial masses are determined using a fitness function.

In other words, each mass presents a solution, and the algorithm is navigated by properly adjusting the gravitational and inertia masses. By lapse of time, we expect that masses be attracted by the heaviest mass. This mass will present an optimum solution in the search space.

The GSA could be considered as an isolated system of masses. It is like a small artificial world of masses obeying the Newtonian laws of gravitation and motion. More precisely, masses obey the following laws:

Law of gravity: each particle attracts every other particle and the gravitational force between two particles is directly proportional to the product of their masses and inversely proportional to *the distance between them*, R. We use here R instead of R^2, because according to our experiment results, R provides better results than R^2 in all experimental cases.
Law of motion: the current velocity of any mass is equal to the sum of the fraction of its previous velocity and the variation in the velocity. Variation in the velocity or acceleration of any mass is equal to the force acted on the system divided by mass of inertia.

Now, consider a system with N agents (masses). We define the position of the ith agent by:

$$X_i = (x_i^1, \ldots, x_i^d, \ldots, x_i^n) \quad \text{for } i = 1, 2, \ldots, N, \tag{6}$$

where x_i^d presents the position of ith agent in the dth dimension.

At a specific time 't', we define the force acting on mass 'i' from mass 'j' as following:

$$F_{ij}^d(t) = G(t)\frac{M_{pi}(t) \times M_{aj}(t)}{R_{ij}(t) + \varepsilon}(x_j^d(t) - x_i^d(t)), \tag{7}$$

where M_{aj} is the active gravitational mass related to agent j, M_{pi} is the passive gravitational mass related to agent i, $G(t)$ is gravitational constant at time t, ε is a small constant, and $R_{ij}(t)$ is the Euclidian distance between two agents i and j:

$$R_{ij}(t) = \|X_i(t), X_j(t)\|_2. \tag{8}$$

To give a stochastic characteristic to our algorithm, we suppose that the total force that acts on agent i in a dimension d be a randomly weighted sum of dth components of the forces exerted from other agents:

$$F_i^d(t) = \sum_{j=1, j\neq i}^{N} rand_j F_{ij}^d(t), \tag{9}$$

where $rand_j$ is a random number in the interval [0,1].

Hence, by the law of motion, the acceleration of the agent i at time t, and in direction dth, $a_i^d(t)$, is given as follows:

$$a_i^d(t) = \frac{F_i^d(t)}{M_{ii}(t)}, \tag{10}$$

where M_{ii} is the inertial mass of ith agent.

Furthermore, the next velocity of an agent is considered as a fraction of its current velocity added to its acceleration. Therefore, its position and its velocity could be calculated as follows:

$$v_i^d(t+1) = rand_i \times v_i^d(t) + a_i^d(t), \tag{11}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1), \tag{12}$$

where $rand_i$ is a uniform random variable in the interval [0,1]. We use this random number to give a randomized characteristic to the search.

The gravitational constant, G, is initialized at the beginning and will be reduced with time to control the search accuracy. In other words, G is a function of the initial value (G_0) and time (t):

$$G(t) = G(G_0, t). \tag{13}$$

Gravitational and inertia masses are simply calculated by the fitness evaluation. A heavier mass means a more efficient agent. This means that better agents have higher attractions and walk more slowly. Assuming the equality of the gravitational and inertia mass, the values of masses are calculated using the map of fitness. We update the gravitational and inertial masses by the following equations:

$$M_{ai} = M_{pi} = M_{ii} = M_i, \quad i = 1, 2, \ldots, N, \tag{14}$$

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)}, \tag{15}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)}, \tag{16}$$

where $fit_i(t)$ represent the fitness value of the agent i at time t, and, $worst(t)$ and $best(t)$ are defined as follows (for a minimization problem):

$$best(t) = \min_{j \in \{1,\ldots,N\}} fit_j(t), \tag{17}$$

$$worst(t) = \max_{j \in \{1,\ldots,N\}} fit_j(t). \tag{18}$$

It is to be noted that for a maximization problem, Eqs. (17) and (18) are changed to Eqs. (19) and (20), respectively:

$$best(t) = \max_{j \in \{1,\ldots,N\}} fit_j(t), \tag{19}$$

$$worst(t) = \min_{j \in \{1,\ldots,N\}} fit_j(t). \tag{20}$$

One way to perform a good compromise between exploration and exploitation is to reduce the number of agents with lapse of time in Eq. (9). Hence, we propose only a set of agents with bigger mass apply their force to the other. However, we should be careful of using this policy because it may reduce the exploration power and increase the exploitation capability.

We remind that in order to avoid trapping in a local optimum the algorithm must use the exploration at beginning. By lapse of iterations, exploration must fade out and exploitation must fade in. To improve the performance of GSA by controlling exploration and exploitation only the *Kbest* agents will attract the others. *Kbest* is a function of time, with the initial value K_0 at the beginning and decreasing with time. In such a way, at the beginning, all agents apply the force, and as time passes, *Kbest* is decreased linearly and at the end there will be just one agent applying force to the others. Therefore, Eq. (9) could be modified as:

$$F_i^d(t) = \sum_{j \in Kbest, j \neq i} rand_j F_{ij}^d(t), \tag{21}$$

where *Kbest* is the set of first K agents with the best fitness value and biggest mass. The different steps of the proposed algorithm are the followings:

(a) Search space identification.
(b) Randomized initialization.
(c) Fitness evaluation of agents.
(d) Update $G(t)$, $best(t)$, $worst(t)$ and $M_i(t)$ *for* $i = 1,2,\ldots,N$.
(e) Calculation of the total force in different directions.
(f) Calculation of acceleration and velocity.
(g) Updating agents' position.
(h) Repeat steps c to g until the stop criteria is reached.
(i) End.

The principle of GSA is shown in Fig. 2.

To see how the proposed algorithm is efficient some remarks are noted:

- Since each agent could observe the performance of the others, the gravitational force is an information-transferring tool.
- Due to the force that acts on an agent from its neighborhood agents, it can see space around itself.
- A heavy mass has a large effective attraction radius and hence a great intensity of attraction. Therefore, agents with a higher performance have a greater gravitational mass. As a result, the agents tend to move toward the best agent.

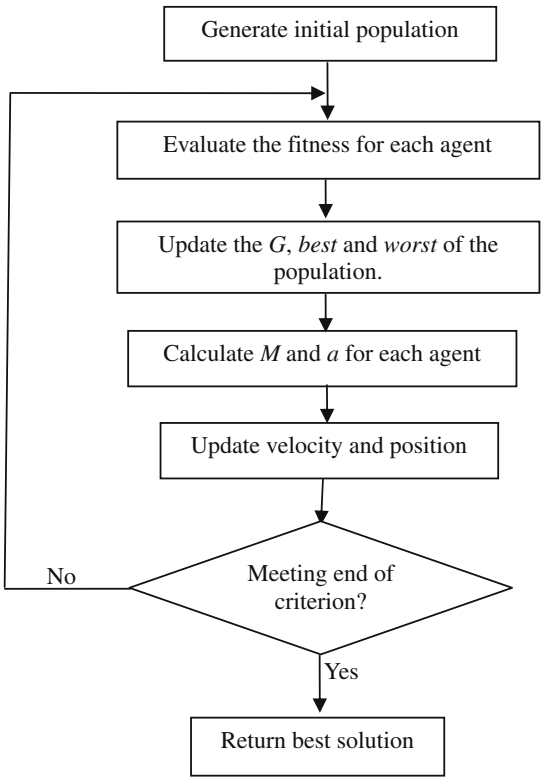

Fig. 2. General principle of GSA.

- The inertia mass is against the motion and make the mass movement slow. Hence, agents with heavy inertia mass move slowly and hence search the space more locally. So, it can be considered as an adaptive learning rate.
- Gravitational constant adjusts the accuracy of the search, so it decreases with time (similar to the temperature in a Simulated Annealing algorithm).
- GSA is a memory-less algorithm. However, it works efficiently like the algorithms with memory. Our experimental results show the good convergence rate of the GSA.
- Here, we assume that the gravitational and the inertia masses are the same. However, for some applications different values for them can be used. A bigger inertia mass provides a slower motion of agents in the search space and hence a more precise search. Conversely, a bigger gravitational mass causes a higher attraction of agents. This permits a faster convergence.

5. Comparative study

In this section, to see how GSA is situated comparing other heuristic search algorithms, we compare theoretically GSA with a stochastic heuristic search, PSO, and a deterministic one, CFO. In addition, an experimental comparison is given in the next section. To start, let us review PSO and CFO algorithms in detail.

5.1. PSO algorithm

PSO is motivated from the simulation of the social behavior (flock of birds). This optimization approach update the population of particles by applying an operator according to the fitness information obtained from the environment so that the individuals of the population can be expected to move towards the better solution. In PSO, x_i^d and v_i^d are calculated as follows [17]:

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1), \tag{22}$$

$$v_i^d(t+1) = w(t)v_i^d(t) + c_1 r_{i1}\left(pbest_i^d - x_i^d(t)\right) + c_2 r_{i2}\left(gbest^d - x_i^d(t)\right), \tag{23}$$

where r_{i1} and r_{i2} are two random variables in the range [0,1], c_1 and c_2 are positive constants, w is the inertia weight. $X_i = (x_i^1, x_i^2, \ldots, x_i^n)$ and $V_i = (v_i^1, v_i^2, \ldots, v_i^n)$ represent position and velocity of the ith particle, respectively. $pbest_i = (pbest_i^1, pbest_i^2, \ldots, pbest_i^n)$ and $gbest = (gbest^1, gbest^2, \ldots, gbest^n)$ represent the best pervious position of the ith particle and the best pervious position among all the particles in the population, respectively.

From Eq. (23), we find that each particle tries to modify its position (X_i) using the distance between the current position and $pbest_i$, and the distance between the current position and $gbest$:

- **GSA versus PSO**
  In both GSA and PSO the optimization is obtained by agents movement in the search space, however the movement strategy is different. Some important differences are as follows:
- In PSO the direction of an agent is calculated using only two best positions, $pbest_i$ and $gbest$. But in GSA, the agent direction is calculated based on the overall force obtained by all other agents.
- In PSO, updating is performed without considering the quality of the solutions, and the fitness values are not important in the updating procedure while in GSA the force is proportional to the fitness value and so the agents see the search space around themselves in the influence of force.
- PSO uses a kind of memory for updating the velocity (due to $pbest_i$ and $gbest$). However, GSA is memory-less and only the current position of the agents plays a role in the updating procedure.
- In PSO, updating is performed without considering the distance between solutions while in GSA the force is reversely proportional to the distance between solutions.
- Finally, note that the search ideas of these algorithms are different. PSO simulates the social behavior of birds and GSA inspires by a physical phenomena.

5.2. CFO algorithm

Central Force Optimization, CFO, is a deterministic multi-dimensional search algorithm. It models the probes that fly through the search space under the influence of gravity [9,10]. At the beginning, the initial probe positions are computed in a deterministic manner. For initialization, the position vector array is filled with a uniform distribution of probes on each coordinate axis at time 0 according to Eq. (24) [9]:

$$\text{for } d = 1 \text{ to } n, \quad \text{for } p = 1 \text{ to } \frac{N}{n}: \quad i = p + \frac{(d-1)N}{n}, \quad x_i^d(0) = x_{\min}^d + \frac{(p-1)(x_{\max}^d - x_{\min}^d)}{\frac{N}{n} - 1}, \tag{24}$$

where n is the dimension of problem and N is the number of probes. In CFO, at each iteration, probes are evaluated and for each probe, M is calculated using fitness function (Eq. (25)). $M_i^{(t)}$ is the mass of the probe i at time t. Then, acceleration is updated using Eq. (26). A new position is calculated using Eq. (27). In this equation, $a_i^d(t)$ and $x_i^d(t)$ are the acceleration and the position of probe i at time t, α and β are two constants. fit_i is the fitness function of probe i that must be maximized:

$$M_i(t) = fit_i, \tag{25}$$

$$a_i^d(t) = G \sum_{j=1, j\neq i}^{N} U(M_j(t) - M_i(t))[M_j(t) - M_i(t)]^{\alpha} \frac{(x_j^d(t) - x_i^d(t))}{R_{ij}(t)^{\beta}}, \tag{26}$$

$$x_i^d(t+1) = x_i^d(t) + \frac{1}{2} a_i^d(t), \tag{27}$$

where G is gravitational constant and $R_{ij}(t)$ is the Euclidean distance between probe i and j at time t, U is the unit step function. As Eqs. (25) and (26) show, in CFO, mass is considered as the difference between values fitness. Since masses can be positive or negative depending on which fitness is greater, the unit step function is included to avoid the possibility of having a negative mass [9]. Due to Eq. (25), mass is dependent proportionally to fitness function. Therefore, in some problems, acceleration may be very high and probes go out of the search space. In CFO any probe that "flew" out of the decision space was returned to the midpoint between its past position and the minimum or maximum value of the coordinate lying outside the allowable range [9].

- **GSA versus CFO**
  In both CFO and GSA the probes positions and accelerations are inspired by particle motion in a gravitational field but they use different formulations.
- One of major differences is that CFO is inherently deterministic and does not use any random parameter in its formulation while GSA is a stochastic search algorithm.
- The acceleration and movement expressions and calculation of the masses in GSA are different from CFO.
- In CFO, the initial probe distribution is systematic (based on a deterministic rule) and has a significant effect on the algorithm's convergence but in GSA the initial distribution is random.
- Another difference is that in CFO, G is a constant while in GSA, G is a control parameter.

6. Experimental results

To evaluate the performance of our algorithm, we applied it to 23 standard benchmark functions [36]. These benchmark functions are presented in Section 6.1. A comparison with Real Genetic Algorithm (RGA) and PSO, is given in Section 6.2. In addition we compare our algorithm with CFO, which is deterministic, in Section 6.3.

6.1. Benchmark functions

Tables 1–3 represent the benchmark functions used in our experimental study. In these tables, n is the dimension of function, f_{opt} is the minimum value of the function, and S is a subset of R^n. The minimum value (f_{opt}) of the functions of Tables 1 and 2 are zero, except for F_8 which has a minimum value of $-418.9829 \times n$. The optimum location (X_{opt}) for functions of Tables 1 and 2, are in $[0]^n$, except for F_5, F_{12} and F_{13} with X_{opt} in $[1]^n$ and F_8 in $[420.96]^n$. A detailed description of the functions of Table 3 is given in Appendix A.

6.2. A comparison with PSO and RGA

We applied GSA to these minimization functions and compared the results with those of RGA as well as PSO. In all cases, population size is set to 50 ($N = 50$). Dimension is 30 ($n = 30$) and maximum iteration is 1000 for functions of Tables 1 and 2, and 500 for functions of Table 3. In PSO, $c_1 = c_2 = 2$ and inertia factor (w) is decreasing linearly from 0.9 to 0.2. In RGA, arith-

Table 1
Unimodal test functions.

Test function	S
$F_1(X) = \sum_{i=1}^{n} x_i^2$	$[-100,100]^n$
$F_2(X) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	$[-10,10]^n$
$F_3(X) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2$	$[-100,100]^n$
$F_4(X) = \max\{\lvert x_i \rvert, 1 \leqslant i \leqslant n\}$	$[-100,100]^n$
$F_5(X) = \sum_{i=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right]$	$[-30,30]^n$
$F_6(X) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$	$[-100,100]^n$
$F_7(X) = \sum_{i=1}^{n} i x_i^4 + random[0,1)$	$[-1.28,1.28]^n$

Table 2
Multimodal test functions.

Test function	S
$F_8(X) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$[-500,500]^n$
$F_9(X) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	$[-5.12,5.12]^n$
$F_{10}(X) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$	$[-32,32]^n$
$F_{11}(X) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600,600]^n$
$F_{12}(x) = \frac{\pi}{n}\{10\sin(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$	$[-50,50]^n$
$y_i = 1 + \frac{x_i + 1}{4}$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i - a)^m & x_i < -a \end{cases}$	
$F_{13}(X) = 0.1\{\sin^2(3\pi x_1) + \sum_{i=1}^{n}(x_i - 1)^2[1 + \sin^2(3\pi x_i + 1)] + (x_n - 1)^2[1 + \sin^2(2\pi x_n)]\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	$[-50,50]^n$

Table 3
Multimodal test functions with fix dimension.

Test function	S
$F_{14}(X) = \left(\frac{1}{500} + \sum_{j=1}^{25}\frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right)^{-1}$	$[-65.53,65.53]^2$
$F_{15}(X) = \sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$	$[-5,5]^4$
$F_{16}(X) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	$[-5,5]^2$
$F_{17}(X) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$	$[-5,10] \times [0,15]$
$F_{18}(X) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	$[-5,5]^2$
$F_{19}(X) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right)$	$[0,1]^3$
$F_{20}(X) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right)$	$[0,1]^6$
$F_{21}(X) = -\sum_{i=1}^{5}[(X - a_i)(X - a_i)^T + c_i]^{-1}$	$[0,10]^4$
$F_{22}(X) = -\sum_{i=1}^{7}[(X - a_i)(X - a_i)^T + c_i]^{-1}$	$[0,10]^4$
$F_{23}(X) = -\sum_{i=1}^{10}[(X - a_i)(X - a_i)^T + c_i]^{-1}$	$[0,10]^4$

metic crossover, Gaussian mutation and roulette wheel selection are used as described in [13]. The crossover and mutation probabilities were set to 0.3 and 0.1, respectively.

In GSA, G is set using Eq. (28), where G_0 is set to 100 and α is set to 20, and T is the total number of iterations (the total age of system):

$$G(t) = G_0 e^{-\alpha \frac{t}{T}}. \tag{28}$$

Furthermore, K_0 is set to N (total number of agents) and is decreased linearly to 1.

We applied the three mentioned algorithms to the benchmark functions, and the results for the following cases are:

Unimodal high-dimensional functions

Functions F_1 to F_7 are unimodal functions. In this case the convergence rate of search algorithm is more important for unimodal functions than the final results because there are other methods which are specifically designed to optimize unimodal functions.

The results are averaged over 30 runs and the average best-so-far solution, average mean fitness function and median of the best solution in the last iteration are reported for unimodal functions in Table 4. As this table illustrates GSA provides better results than RGA and PSO for all functions. The largest difference in performance between them occurs with these

Table 4
Minimization result of benchmark functions in Table 1 with $n = 30$. Maximum number of iterations = 1000.

		RGA	PSO	GSA
F_1	Average best-so-far	23.13	1.8×10^{-3}	7.3×10^{-11}
	Median best-so-far	21.87	1.2×10^{-3}	7.1×10^{-11}
	Average mean fitness	23.45	5.0×10^{-2}	2.1×10^{-10}
F_2	Average best-so-far	1.07	2.0	4.03×10^{-5}
	Median best-so-far	1.13	1.9×10^{-3}	4.07×10^{-5}
	Average mean fitness	1.07	2.0	6.9×10^{-5}
F_3	Average best-so-far	$5.6 \times 10^{+3}$	$4.1 \times 10^{+3}$	$0.16 \times 10^{+3}$
	Median best-so-far	$5.6 \times 10^{+3}$	$2.2 \times 10^{+3}$	$0.15 \times 10^{+3}$
	Average mean fitness	$5.6 \times 10^{+3}$	$2.9 \times 10^{+3}$	$0.16 \times 10^{+3}$
F_4	Average best-so-far	11.78	8.1	3.7×10^{-6}
	Median best-so-far	11.94	7.4	3.7×10^{-6}
	Average mean fitness	11.78	23.6	8.5×10^{-6}
F_5	Average best-so-far	$1.1 \times 10^{+3}$	$3.6 \times 10^{+4}$	25.16
	Median best-so-far	$1.0 \times 10^{+3}$	$1.7 \times 10^{+3}$	25.18
	Average mean fitness	$1.1 \times 10^{+3}$	$3.7 \times 10^{+4}$	25.16
F_6	Average best-so-far	24.01	1.0×10^{-3}	8.3×10^{-11}
	Median best-so-far	24.55	6.6×10^{-3}	7.7×10^{-11}
	Average mean fitness	24.52	0.02	2.6×10^{-10}
F_7	Average best-so-far	0.06	0.04	0.018
	Median best-so-far	0.06	0.04	0.015
	Average mean fitness	0.56	1.04	0.533

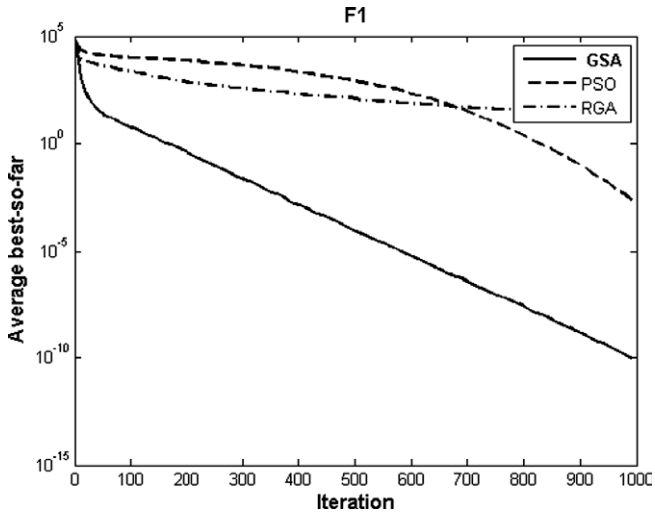

Fig. 3. Comparison of performance of GSA, PSO and RGA for minimization of F_1 with $n = 30$.

unimodal functions. This is due to the attractive power of GSA. Also, the good convergence rate of GSA could be concluded from Figs. 3 and 4. According to these figures, GSA tends to find the global optimum faster than other algorithms and hence has a higher convergence rate.

Multimodal high-dimensional functions

Multimodal functions have many local minima and almost are most difficult to optimize. For multimodal functions, the final results are more important since they reflect the ability of the algorithm in escaping from poor local optima and locating a near-global optimum. We have carried out experiments on F_8 to F_{13} where the number of local minima increases exponentially as the dimension of the function increases. The dimension of these functions is set to 30. The results are averaged over 30 runs and the average best-so-far solution, average mean fitness function and median of the best solution in the last iteration are reported for these functions in Table 5. For F_{10}, F_{12} and F_{13}, GSA performs much better solution than the others. However, for function F_8, GSA cannot tune itself and have not a good performance. The progress of the average best-so-far solution over 30 run for functions F_{10} and F_{12} are shown in Figs. 5 and 6.

Multimodal low-dimensional functions

Table 6 shows the comparison between GSA, RGA and PSO on multimodal low-dimensional benchmark functions of Table 3. The results show that RGA, PSO and GSA have similar solutions and the performances are almost the same as Figs. 7 and 8 confirm it.

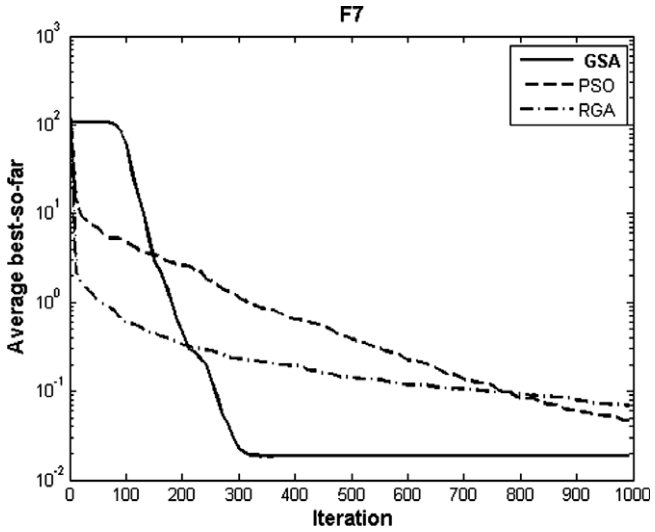

Fig. 4. Comparison of performance of GSA, PSO and RGA for minimization of F_7 with $n = 30$.

Table 5
Minimization result of benchmark functions in Table 2 with $n = 30$. Maximum number of iterations = 1000.

		RGA	PSO	GSA
F_8	Average best-so-far	$-1.2 \times 10^{+4}$	$-9.8 \times 10^{+3}$	$-2.8 \times 10^{+3}$
	Median best-so-far	$-1.2 \times 10^{+4}$	$-9.8 \times 10^{+3}$	$-2.6 \times 10^{+3}$
	Average mean fitness	$-1.2 \times 10^{+4}$	$-9.8 \times 10^{+3}$	$-1.1 \times 10^{+3}$
F_9	Average best-so-far	5.90	55.1	15.32
	Median best-so-far	5.71	56.6	14.42
	Average mean fitness	5.92	72.8	15.32
F_{10}	Average best-so-far	2.13	9.0×10^{-3}	6.9×10^{-6}
	Median best-so-far	2.16	6.0×10^{-3}	6.9×10^{-6}
	Average mean fitness	2.15	0.02	1.1×10^{-5}
F_{11}	Average best-so-far	1.16	0.01	0.29
	Median best-so-far	1.14	0.0081	0.04
	Average mean fitness	1.16	0.055	0.29
F_{12}	Average best-so-far	0.051	0.29	0.01
	Median best-so-far	0.039	0.11	4.2×10^{-13}
	Average mean fitness	0.053	$9.3 \times 10^{+3}$	0.01
F_{13}	Average best-so-far	0.081	3.1×10^{-18}	3.2×10^{-32}
	Median best-so-far	0.032	2.2×10^{-23}	2.3×10^{-32}
	Average mean fitness	0.081	$4.8 \times 10^{+5}$	3.2×10^{-32}

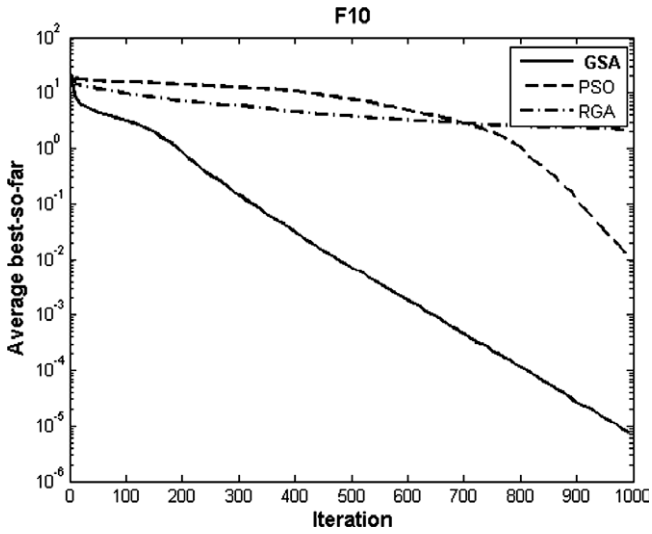

Fig. 5. Comparison of performance of GSA, PSO and RGA for minimization of F_{10} with $n = 30$.

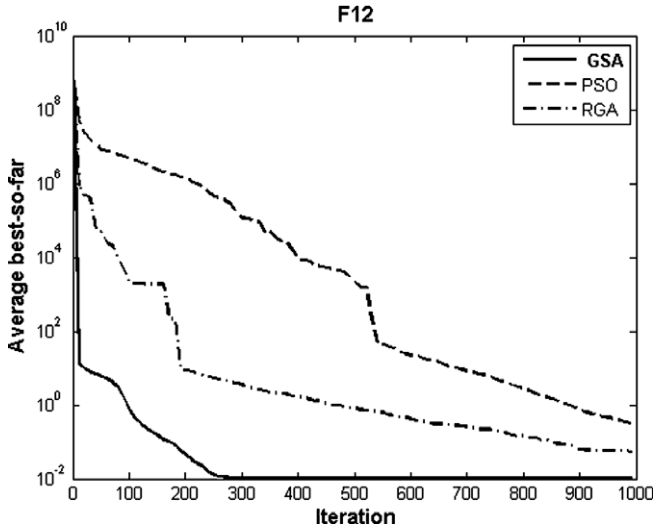

Fig. 6. Comparison of performance of GSA, PSO and RGA for minimization of F_{12} with $n = 30$.

6.3. Comparison with CFO

Based on the theoretical comparison of GSA and CFO, one could understand that it is very difficult to compare the two algorithms in the same conditions. CFO is a deterministic search algorithm and its performance is highly dependent on the initial population generation and the size of the population. Specially, when the complexity and the dimension of the problem at hand increases, CFO will be more sensitive to the population size and initialization criterion. Moreover, in this condition to obtain acceptable results, CFO must start with a large size of population. This means that before solving a problem using CFO, we have to know some a priori information about its complexity to set up the number of population or run the algorithm many times to attain a suitable population value with try and error. However, GSA is a stochastic search algorithm and can optimize a wide range of problems with a fixed small size population.

Due to the above-mentioned reasons, it is not possible to compare GSA and CFO in optimization of high-dimensional functions with the same condition. Therefore, we compared CFO and GSA in low-dimensional problems. Since CFO searches for maxima, the negative of the functions is tested for CFO. In addition, according to [9], in order to avoid any bias resulting from the locations of maxima relative to the initial probe distribution, the maxima were offset if necessary. In CFO, the position vector array is filled with uniform distribution of probes on each coordinate axis at step 0. In GSA, initialization is random. The number of agents and the maximum number of iterations are set to 60 and 500, respectively. We use 60 agents

because CFO needs a large population size. The parameters of GSA are set as the previous section. For CFO we have $G = 2$, $\alpha = 2$ and $\beta = 2$ [9]. Note that we compare both GSA and CFO only for the functions that reported by Formato [9], because we need some a priori information about the function to be optimized to apply to CFO.

Results of GSA are averaged over 30 runs. CFO is deterministic and hence has the same results for each run. The results are given in Table 7, where we have also reported the result of CFO with random initialization (not uniform distribution initial-

Table 6
Minimization result of benchmark functions in Table 3. Maximum number of iterations = 500.

		RGA	PSO	GSA
F_{14} $n = 2$	Average best-so-far	0.998	0.998	3.70
	Median best-so-far	0.998	0.998	2.07
	Average mean fitness	0.998	0.998	9.177
F_{15} $n = 4$	Average best-so-far	4.0×10^{-3}	2.8×10^{-3}	8.0×10^{-3}
	Median best-so-far	1.7×10^{-3}	7.1×10^{-4}	7.4×10^{-4}
	Average mean fitness	4.0×10^{-3}	215.60	9.0×10^{-3}
F_{16} $n = 2$	Average best-so-far	−1.0313	−1.0316	−1.0316
	Median best-so-far	−1.0315	−1.0316	−1.0316
	Average mean fitness	−1.0313	−1.0316	−1.0316
F_{17} $n = 2$	Average best-so-far	0.3996	0.3979	0.3979
	Median best-so-far	0.3980	0.3979	0.3979
	Average mean fitness	0.3996	2.4112	0.3979
F_{18} $n = 2$	Average best-so-far	5.70	3.0	3.0
	Median best-so-far	3.0	3.0	3.0
	Average mean fitness	5.70	3.0	3.0
F_{19} $n = 3$	Average best-so-far	−3.8627	−3.8628	−3.7357
	Median best-so-far	−3.8628	−3.8628	−3.8628
	Average mean fitness	−3.8627	−3.8628	−3.8020
F_{20} $n = 6$	Average best-so-far	−3.3099	−3.2369	−2.0569
	Median best-so-far	−3.3217	−3.2031	−1.9946
	Average mean fitness	−3.3098	−3.2369	−1.6014
F_{21} $n = 4$	Average best-so-far	−5.6605	−6.6290	−6.0748
	Median best-so-far	−2.6824	−5.1008	−5.0552
	Average mean fitness	−5.6605	−5.7496	−6.0748
F_{22} $n = 4$	Average best-so-far	−7.3421	−9.1118	−9.3399
	Median best-so-far	−10.3932	−10.402	−10.402
	Average mean fitness	−7.3421	−9.9305	−9.3399
F_{23} $n = 4$	Average best-so-far	−6.2541	−9.7634	−9.4548
	Median best-so-far	−4.5054	−10.536	−10.536
	Average mean fitness	−6.2541	−8.7626	−9.4548

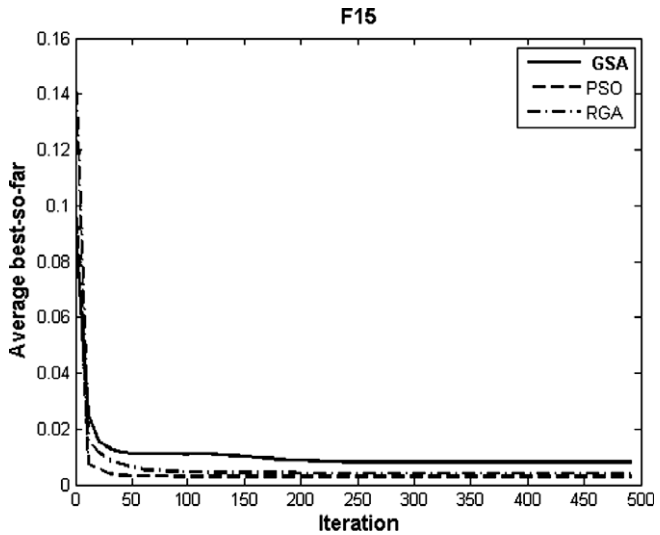

Fig. 7. Comparison of performance of GSA, PSO and RGA for minimization of F_{15} with $n = 4$.

ization) in the last row. These results are averaged over 30 runs. As Table 7 shows, CFO does not present good results with random initialization and the initialization criterion has a significant effect on the obtained results. Performance of each algorithm is shown in Figs. 9–11 for three functions. The results show that GSA has better solution than CFO except for functions F_5, F_8 and F_{11}.

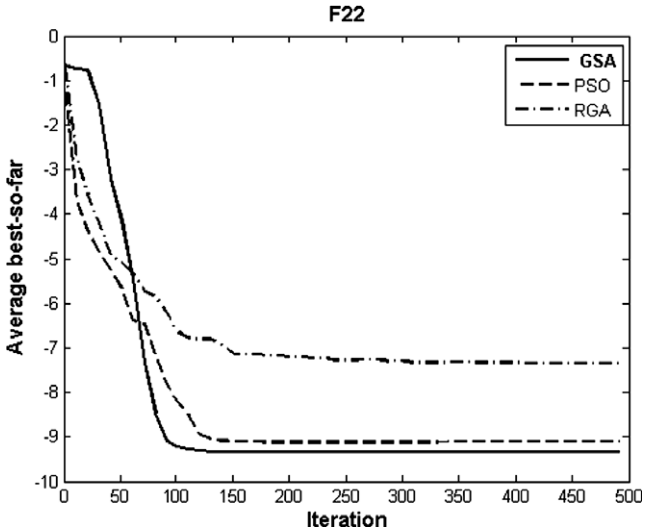

Fig. 8. Comparison of performance of GSA, PSO and RGA for minimization of F_{22} with $n = 4$.

Table 7
Minimization result of some functions in Tables 1–3. Maximum number of iterations = 500. Results for GSA and CFO with random initialization are averaged over 30 runs.

	F_1	F_5	F_6	F_8	F_9	F_{10}	F_{11}	F_{14}	F_{16}	F_{17}
n (dimension)	5	5	5	5	5	5	5	2	2	2
GSA	$\mathbf{3.1 \times 10^{-19}}$	1.62	$\mathbf{3.3 \times 10^{-19}}$	$-1.08 \times 10^{+3}$	**0.99**	$\mathbf{1.1 \times 10^{-9}}$	0.61	**2.87**	**−1.031**	**0.39**
CFO	1.13	**0.02**	0.093	$-1.37 \times 10^{+3}$	1.09	1.13	**0.02**	3.18	−1.025	0.41
CFO with random initialization	880.18	$1.22 \times 10^{+5}$	$1.03 \times 10^{+3}$	$\mathbf{-1.94 \times 10^{+3}}$	23.52	2.81	9.46	8.88	−0.98	0.46

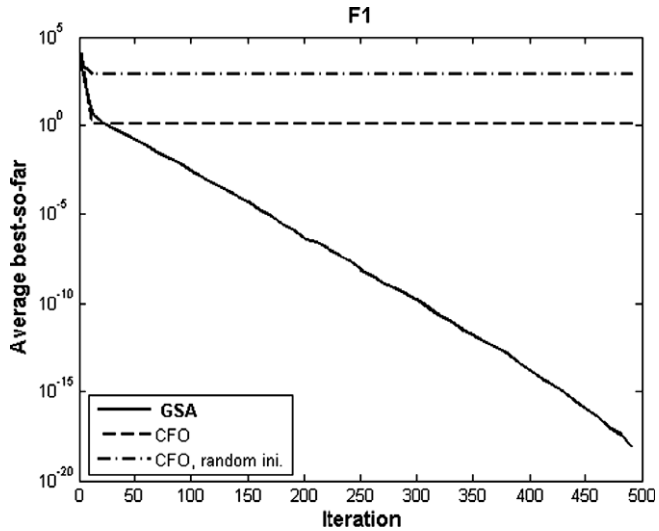

Fig. 9. Comparison of performance of GSA, CFO and CFO with random initialization for minimization of F_1 with $n = 5$.

For optimization of high-dimensional functions, CFO should run with large amounts of probes. Due to the deterministic characteristics of CFO, it converges in the first few iterations. The simulations of some functions with 30 dimensions have been reported in [9].

The reported results for dimension 30 by [9] with different number of agents (N), are summarized in Table 8. By comparing these results with average best-so-far of GSA in Tables 4–6, we can see that GSA provides better solutions except for F_5, F_{11} and F_{14}. CFO converges at the first few iterations and cannot tune itself in the local optimum.

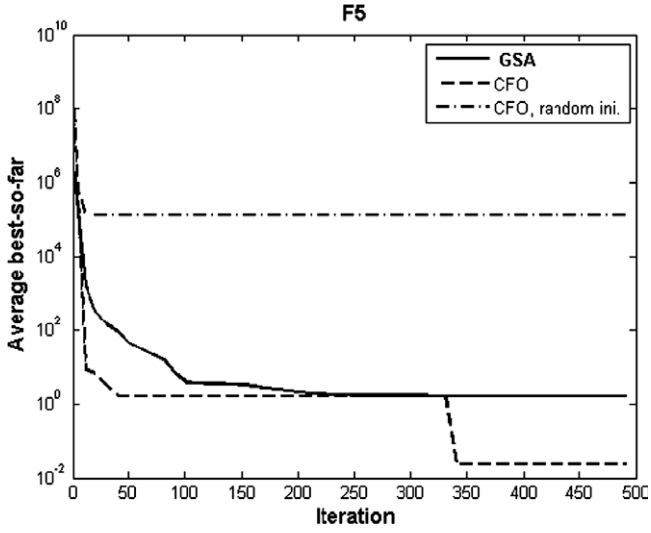

Fig. 10. Comparison of performance of GSA, CFO and CFO with random initialization for minimization of F_5 with $n = 5$.

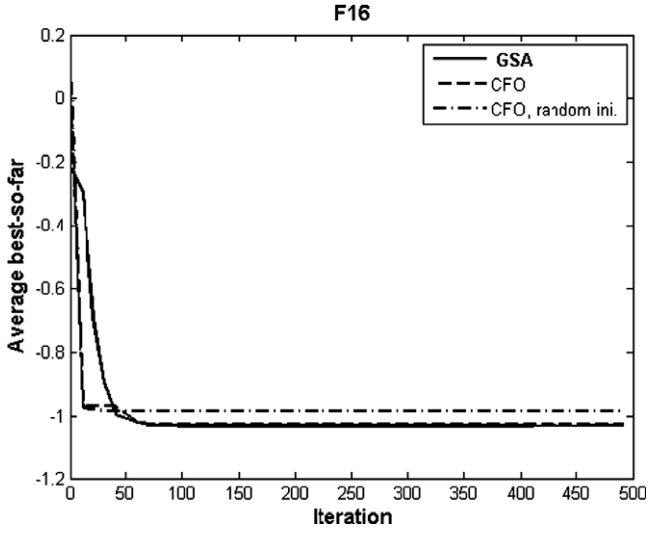

Fig. 11. Comparison of performance of GSA, CFO and CFO with random initialization for minimization of F_{16} with $n = 2$.

Table 8
Optimization result of some functions of Tables 1–3 with CFO reported in [9]. Negative of results are reported.

	F_1	F_5	F_6	F_8	F_9	F_{10}	F_{11}	F_{14}	F_{16}	F_{17}
n	30	30	30	30	30	30	30	2	2	2
N [9]	15000	60	600	240	600	780	780	240	260	400
Iterations [9]	2	250	4	8	8	5	6	2	16	18
Best so far [9]	0.0836	3.8	1	−12569.1	30.53	1	0.045	1.2	−1.027	0.398

7. Conclusion

In recent years, various Heuristic optimization methods have been developed. Some of these algorithms are inspired by swarm behaviors in nature. In this article, a new optimization algorithm that is called Gravitational Search Algorithm (GSA), is introduced. GSA is constructed based on the law of Gravity and the notion of mass interactions. The GSA algorithm uses the theory of Newtonian physics and its searcher agents are the collection of masses. In GSA, we have an isolated system of masses. Using the gravitational force, every mass in the system can see the situation of other masses. The gravitational force is therefore a way of transferring information between different masses.

In order to evaluate our algorithm, we have examined it on a set of various standard benchmark functions. The results obtained by GSA in most cases provide superior results and in all cases are comparable with PSO, RGA and CFO. Finally it must be noted that a local search – as used in [12] – after found result may be beneficial.

Acknowledgements

The authors would like to thank the INS Editorial Board and the anonymous reviewers for their very helpful suggestions. In addition, the authors give kind respect and special thanks to Dr. Richard Formato for proposing some useful references and giving us his CFO programs. Also, the authors would like to extend their appreciation to Dr. Saeid Seydnejad for proof reading the manuscript and providing valuable comments.

Appendix A

See Tables A.1–A.8

Table A.1
a_{ij} in F_{14}.

$(a_{ij}) = \begin{pmatrix} -32,-16,0,16,32,-32,\ldots,0,16,32 \\ -32,-32,-32,-32,-16,\ldots,32,32,32 \end{pmatrix}$

Table A.2
a_i and b_i in F_{15}.

i	1	2	3	4	5	6	7	8	9	10	11
a_i	0.1957	0.1947	0.1735	0.1600	0.0844	0.0627	0.0456	0.0342	0.0342	0.0235	0.0246
b_i^{-1}	0.25	0.5	1	2	4	6	8	10	12	14	16

Table A.3
a_{ij} and c_i in F_{19}.

i	a_{ij}, $j = 1,2,3$			c_i
1	3	10	30	1
2	0.1	10	35	1.2
3	3	10	30	3
4	0.1	10	30	3.2

Table A.4
P_{ij} in F_{19}.

i	P_{ij}, $j = 1,2,3$		
1	0.3689	0.1170	0.2673
2	0.4699	0.4387	0.7470
3	0.1091	0.8732	0.5547
4	0.03815	0.5743	0.8828

Table A.5
a_{ij} and c_i in F_{20}.

i	a_{ij}, $j = 1,2,3,4,5,6$						c_i
1	10	3	17	3.5	1.7	8	1
2	0.05	10	17	0.1	8	14	1.2
3	3	3.5	1.7	10	17	8	3
4	17	8	0.05	10	0.1	14	3.2

Table A.6
P_{ij} in F_{20}.

i	P_{ij}, $j = 1,2,3,4,5,6$					
1	0.131	0.169	0.556	0.012	0.828	0.588
2	0.232	0.413	0.830	0.373	0.100	0.999
3	0.234	0.141	0.352	0.288	0.304	0.665
4	0.404	0.882	0.873	0.574	0.109	0.038

Table A.7
a_{ij} and c_i in F_{21}, F_{22} and F_{23}.

i	a_{ij}, $j = 1,2,3,4$				c_i
1	4	4	4	4	0.1
2	1	1	1	1	0.2
3	8	8	8	8	0.2
4	6	6	6	6	0.4
5	3	7	3	7	0.4
6	2	9	2	9	0.6
7	5	5	3	3	0.3
8	8	1	8	1	0.7
9	6	2	6	2	0.5
10	7	3.6	7	3.6	0.5

Table A.8
Optima in functions of Table 3.

F	X_{opt}	f_{opt}
F_{14}	$(-32, 32)$	1
F_{15}	$(0.1928, 0.1908, 0.1231, 0.1358)$	0.00030
F_{16}	$(0.089, -0.712)$, $(-0.089, 0.712)$	−1.0316
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} $i = 1, \ldots, 5$	−10.1532
F_{22}	7 local minima in a_{ij} $i = 1, \ldots, 7$	−10.4028
F_{23}	10 local minima in a_{ij} $i = 1, \ldots, 10$	−10.5363

References

[1] A. Badr, A. Fahmy, A proof of convergence for ant algorithms, Information Sciences 160 (2004) 267–279.
[2] Z. Baojiang, L. Shiyong, Ant colony optimization algorithm and its application to neuro-fuzzy controller design, Journal of Systems Engineering and Electronics 18 (2007) 603–610.
[3] F.V.D. Bergh, A.P. Engelbrecht, A study of particle swarm optimization particle trajectories, Information Sciences 176 (2006) 937–971.
[4] O. Cordon, S. Damas, J. Santamarı, A fast and accurate approach for 3D image registration using the scatter search evolutionary algorithm, Pattern Recognition Letters 27 (2006) 1191–1200.
[5] M. Dorigo, V. Maniezzo, A. Colorni, The ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man, and Cybernetics – Part B 26 (1) (1996) 29–41.
[6] W. Du, B. Li, Multi-strategy ensemble particle swarm optimization for dynamic optimization, Information Sciences 178 (2008) 3096–3109.
[7] I. Ellabib, P. Calamai, O. Basir, Exchange strategies for multiple ant colony system, Information Sciences 177 (2007) 1248–1264.
[8] J.D. Farmer, N.H. Packard, A.S. Perelson, The immune system, adaptation and machine learning, Physica D 2 (1986) 187–204.
[9] R.A. Formato, Central force optimization: a new metaheuristic with applications in applied electromagnetics, Progress in Electromagnetics Research 77 (2007) 425–491.
[10] R.A. Formato, Central force optimization: a new nature inspired computational framework for multidimensional search and optimization, Studies in Computational Intelligence 129 (2008) 221–238.
[11] V. Gazi, K.M. Passino, Stability analysis of social foraging swarms, IEEE Transactions on Systems, Man, and Cybernetics – Part B 34 (1) (2004) 539–557.
[12] C. Hamzaçebi, Improving genetic algorithms' performance by local search for continuous function optimization, Applied Mathematics and Computation 196 (1) (2008) 309–317.
[13] R.L. Haupt, E. Haupt, Practical Genetic Algorithms, second ed., John Wiley & Sons, 2004.
[14] D. Holliday, R. Resnick, J. Walker, Fundamentals of physics, John Wiley and Sons, 1993.
[15] A. Kalinlia, N. Karabogab, Artificial immune algorithm for IIR filter design, Engineering Applications of Artificial Intelligence 18 (2005) 919–929.
[16] C. Karakuzu, Fuzzy controller training using particle swarm optimization for nonlinear system control, ISA Transactions 47 (2) (2008) 229–239.
[17] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948.
[18] I.R. Kenyon, General Relativity, Oxford University Press, 1990.
[19] D.H. Kim, A. Abraham, J.H. Cho, A hybrid genetic algorithm and bacterial foraging approach for global optimization, Information Sciences 177 (2007) 3918–3937.
[20] T.H. Kim, I. Maruta, T. Sugie, Robust PID controller tuning based on the constrained particle swarm optimization, Automatica 44 (2008) 1104–1110.
[21] S. Kirkpatrick, C.D. Gelatto, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[22] A. Lazar, R.G. Reynolds, Heuristic knowledge discovery for archaeological data using genetic algorithms and rough sets, Artificial Intelligence Laboratory, Department of Computer Science, Wayne State University, 2003.
[23] Y.L. Lin, W.D. Chang, J.G. Hsieh, A particle swarm optimization approach to nonlinear rational filter modeling, Expert Systems with Applications 34 (2008) 1194–1199.
[24] Y. Liu, Z. Yi, H. Wu, M. Ye, K. Chen, A tabu search approach for the minimum sum-of-squares clustering problem, Information Sciences 178 (2008) 2680–2704.
[25] M. Lozano, F. Herrera, J.R. Cano, Replacement strategies to preserve useful diversity in steady-state genetic algorithms, Information Sciences 178 (2008) 4421–4433.
[26] R. Mansouri, F. Nasseri, M. Khorrami, Effective time variation of G in a model universe with variable space dimension, Physics Letters 259 (1999) 194–200.
[27] H. Nezamabadi-pour, S. Saryazdi, E. Rashedi, Edge detection using ant algorithms, Soft Computing 10 (2006) 623–628.
[28] E. Rashedi, Gravitational Search Algorithm, M.Sc. Thesis, Shahid Bahonar University of Kerman, Kerman, Iran, 2007 (in Farsi).
[29] S.J. Russell, P. Norvig, Artificial Intelligence a Modern Approach, Prentice Hall, Upper Saddle River, New Jersy, 1995.
[30] B. Schutz, Gravity from the Ground Up, Cambridge University Press, 2003.
[31] X. Tan, B. Bhanu, Fingerprint matching by genetic algorithms, Pattern Recognition 39 (2006) 465–477.
[32] K.S. Tang, K.F. Man, S. Kwong, Q. He, Genetic algorithms and their applications, IEEE Signal Processing Magazine 13 (6) (1996) 22–37.
[33] P. Tarasewich, P.R. McMullen, Swarm intelligence: power in numbers, Communication of ACM 45 (2002) 62–67.
[34] P.K. Tripathi, S. Bandyopadhyay, S.K. Pal, Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients, Information Sciences 177 (2007) 5033–5049.
[35] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1997) 67–82.
[36] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Transactions on Evolutionary Computation 3 (1999) 82–102.