Proceedings of the Estonian Academy of Sciences,
2016, **65**, 1, 15–27
doi: 10.3176/proc.2016.1.01
Available online at www.eap.ee/proceedings

COMPUTER
SCIENCE

Modified particle swarm optimization algorithm based on gravitational field interactions

Margarita Spichakova

Institute of Cybernetics at Tallinn University of Technology, Akadeemia tee 21, 12618 Tallinn, Estonia; margo@cs.ioc.ee

Received 9 February 2015, revised 11 May 2015, accepted 25 May 2015, available online 10 December 2015

Abstract. In this paper we present the modified particle swarm optimization algorithm, where gravitational interactions between particles are used for computing learning coefficients. The behaviour of the algorithm is demonstrated by solving the two-dimensional Diophantine equation problem. This allows us to observe the search space and workflow of the algorithm directly on the two-dimensional plane.

Key words: particle swarm optimization, Diophantine equation solver, gravitationally inspired heuristic search.

1. INTRODUCTION

Define the *search problem* as finding an optimum (minimum or maximum) of some given function. The set of *points*, presenting the function arguments, gives us the *search space*. For each point in the search space there is the value of the function. The task is to find the point (argument), which gives the optimal value of the function.

If no information about the search space is available, two main opportunities exist: (1) to exhaustively traverse the search space or (2) to choose some random points and pick up the most suitable one. If the search space is infinite, we define some *range*. Traversing the whole search space takes time, but gives an exact result. In the case of an infinite search space, the results also depend on the definition of the range, and it may happen that the range does not contain the optimum. Randomly generating a small number of points in the search space gives a solution very fast, but the quality of the solution is questionable. The stochastic optimization presents the compromise between exhaustive search and choosing random points.

Optimization techniques include several approaches, some of which are *deterministic* procedures and others contain randomness and probabilistic computations. The main advantage of *stochastic optimization* is that it can be applied to any search problem without specific knowledge about the structure of the search space. Stochastic optimization may also be helpful when the complexity of deterministic methods grows rapidly with the search space size.

Two main properties must be implemented in any stochastic optimization method: the *exploration* and the *exploitation*. The exploration is the ability of the method to explore the entire search space in a global way and the exploitation is its ability to focus in the local area and search for a more exact solution.

Stochastic optimization methods have several ideas in common:

- The *search space* is defined as a set of points where each point represents a *candidate solution*. Usually, candidate solutions are presented indirectly, by some structure, which encodes the candidate solution. Initially, some fixed number of points are generated randomly.

- By using the *evaluation function* we can assign the *score value*, which shows how good a solution is.
- The search algorithm contains *modification operators* that allow the construction of new solutions from the existing ones.

The nature has been the source of inspiration for constructing new stochastic search algorithms. There is a set of methods, such as evolutionary algorithms, genetic algorithms, evolutionary programming, which are based on the theory of evolution. Some methods simulate social behaviour, for example, *particle swarm optimization (PSO)* imitates the social behaviour of birds, ant colony optimization applies ideas of the behaviour of ants foraging for food. Some stochastic optimization methods are based on the laws of physics, for example, simulated annealing is based on the thermodynamic effect. Others are based on gravitational force. For example, *Central Force Optimization* is a deterministic gravity-based search algorithm, which simulates the group of probes [2]. *Space Gravitational Optimization* [3] simulates asteroids flying through a curved search space. A gravitationally-inspired variation of local search, the *Gravitational Emulation Local Search Algorithm*, was proposed by Webster [14] and Webster and Bernhard [15]. The newest one, the *Gravitational Search Algorithm*, was proposed by Rashedi et al. [10–12] as a stochastic variation of Central Force Optimization. A discrete modification of the Gravitational Search Algorithm was proposed by Zibanezhad et al. [16] in the context of Web-Service composition.

In this article we discuss the family of PSO algorithms. Standard PSO contains three parameters to control the algorithm. Traditionally, the values of the parameters are defined by end users. In general, no exact methods exist for defining the parameters, so, usually, it is done empirically. Also, it is possible to define a new search problem for finding values of parameters and applying stochastic optimization. This process is called *meta-heuristics*.

We present a modification of the PSO algorithm based on gravitational interactions (GI) between particles (PSO + GI), which automatically adjusts the parameters of the PSO algorithm. The proposed method solves the parameter adjusting problem by replacing parameters with computed values.

Both methods, the standard PSO algorithm and the proposed modification PSO + GI algorithm, are tested on the Diophantine equation solver task (see Table 1 in Section 5.1 for test equations). This search problem is chosen for illustrative purposes.

The paper is organized as follows. Section 2 defines the search problem, Section 3 presents the standard PSO algorithm, and Section 4 describes the new method PSO + GI. Section 5 covers the behaviour of the proposed method and Section 6 contains the conclusion and future plans.

2. DIOPHANTINE EQUATION SOLVER

A *Diophantine equation (DE)* has the form

$$F(x_1, x_2, \ldots, x_m) = 0, \tag{1}$$

where coefficients and variables are integers and F can be considered as a polynomial function.

For simple cases there are deterministic methods for solving such equations. For example, in the case of a linear DE with two variables, the solution can be found by using the Euclidean algorithm for the greatest common divider.

The problem of finding a general deterministic method for solving any DE is known as ‘Hilbert’s tenth problem’. It was proven by Y. Matiyasevich in 1970 that there is no such deterministic method. Therefore, the use of stochastic optimization methods is helpful.

Several stochastic optimization algorithms are used for solving DEs. Abraham [1] applied the standard PSO algorithm to the DEs of simpler forms (see Eq. 2) and tested it on two sets: the first set – DEs with n from 2 to 15 (see Eq. 2) and the second one with equations with power 2 with 2 to 12 variables. In all cases the PSO method was able to find the solutions.

The genetic algorithm can also be used for solving a DE. For example, there is a genetic algorithm tutorial [4], where the genetic algorithm is demonstrated on solving the equation $a + 2b + 3c + 4d = 30$.

To use the stochastic optimization method for the DE solver problem, we need to define at least three things: the search problem, search space, and evaluation function.

The dimensionality of the search space depends on the number of arguments to be found. The restriction $m = 2$ gives us only two variables x and y, so the general DE has the form

$$a \cdot x^n + b \cdot y^n = d. \tag{2}$$

The test equations are presented in Table 1 in Section 5.1.

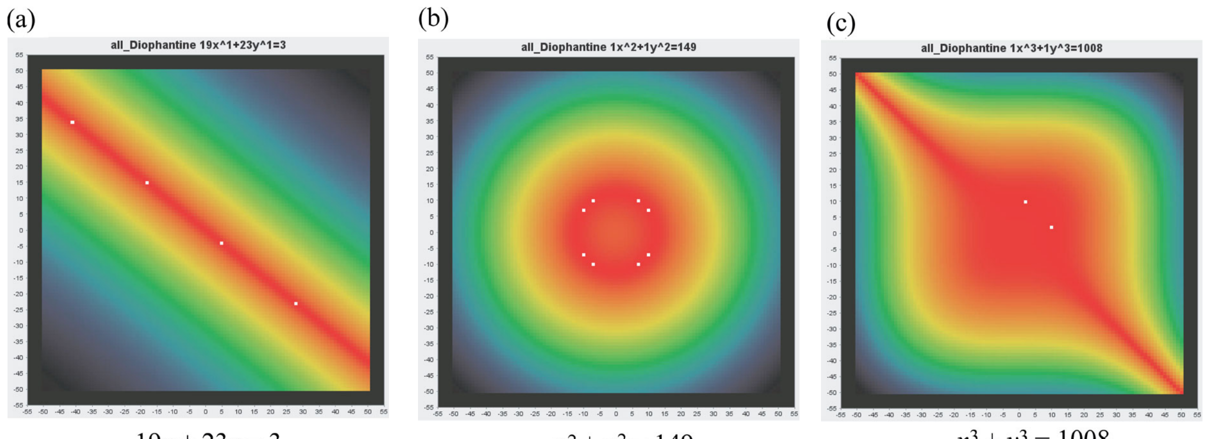
The *evaluation function* must return the value for each point (candidate solution) in the search space. In our case, the candidate solution is a pair (x_p, y_p). In the easiest case, we can try a point of the search space (x_p, y_p) as a solution of the DE and get the result *true* or *false*. Unfortunately, this gives not much information about how close this point was to optima.

We can define the evaluation function as a distance between two points:

$$\begin{aligned} a \cdot x^n + b \cdot y^n &= d, \\ a \cdot x_p^n + b \cdot y_p^n &= dd, \\ f &= |dd - d|. \end{aligned} \tag{3}$$

If we apply our candidate solution (x_p, y_p) to the equation, we can compute dd. In our original Eq. 2, this value is equal to d. If $dd = d$, then (x_p, y_p) is our optimal solution. To measure how far the candidate solution is from the unknown optimum, we define the distance f, which will be our evaluation function. So, the search problem is now defined as *minimize the evaluation function* f. The point (x_p, y_p) will be considered as an optimal solution if $f = 0$.

As we mentioned before, the search space is defined as a set of candidate solutions (x_p, y_p). We also restricted the search area by upper and lower bounds, which are defined by the user. The coordinates (x_p, y_p) present two dimensions and the evaluation function value gives the third one. To illustrate such a search space, we will use the diagram where points with coordinates correspond to candidate solutions and the evaluation function value is coded by colour in such a way that the colour ranges from blue (that corresponds to long distances) to red (that corresponds to smaller distances), and the optimal values are presented by white colour. Examples of such diagrams are shown in Fig. 1.



$19x + 23y = 3$ $x^2 + y^2 = 149$ $x^3 + y^3 = 1008$

Fig. 1. Examples of the search space.

3. PARTICLE SWARM OPTIMIZATION

The PSO algorithm is inspired by social behaviour of some set of objects, for example, bird flock or fish school [5]. The general idea of this method can be described as follows. The search space contains a set of points and each point has its value, which is assigned by the *evaluation function*. The *task* is to find the optima of this function. There is also a set of particles that are moving on the defined search space. The movement laws can be considered as interactions between objects. Using those movement laws, the objects must find the positions with better values or even optima.

3.1. Standard PSO algorithm

Consider a set of particles – a *swarm*. Each particle is characterized by the *position vector*, *velocity vector*, and the best known position for this object. There exists also the global best known position for the whole swarm.

The *position* vector $p_d, d \in [0 \ldots n]$ presents a *candidate solution*. The dimensionality n of the vector depends on the problem size. We assign the value for each candidate solution using the evaluation function. According to those values we can choose the global best known position *Gbest*, which is the point with the optimal value found so far by the whole swarm, and the local best known position *Pbest*, which is the best position that was found by this exact particle.

The *velocity* vector $v_d, d \in [0 \ldots n]$ represents the trend of movement of the particle. It is computed by using the equation

$$v_d(t) = \alpha \cdot v_d(t-1) + \beta \cdot r_1 \cdot (Pbest_d - p_d(t-1)) + \gamma \cdot r_2 \cdot (Gbest_d - p_d(t-1)), \tag{4}$$

where

- α, β, γ are learning coefficients with α representing the inertia, β is the cognitive memory, and γ is the social memory; those coefficients must be defined by the user;
- r_1 and r_2 are random values in the range $[0 \ldots 1]$;
- $Pbest_d$ is the local best known position for the particle and $Gbest_d$ is the global best known position of the swarm;
- $v_d(t)$ is the new value of the velocity vector at dimension d and $v_d(t-1)$ is the previous value of the velocity.

The new position $p_d(t)$ is simply defined as the sum of the previous position $p_d(t-1)$ and the new velocity $v_d(t)$:

$$p_d(t) = p_d(t-1) + v_d(t). \tag{5}$$

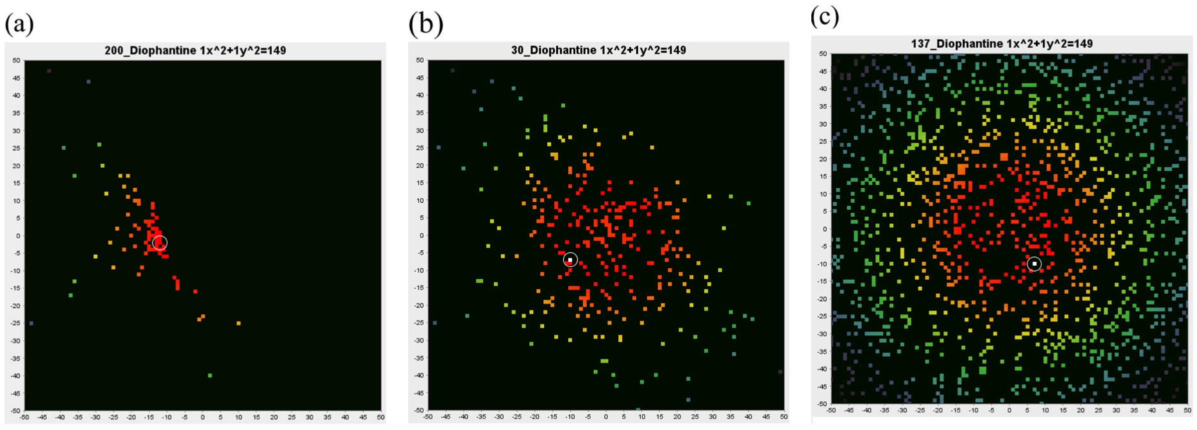
The PSO algorithm is presented in Algorithm 1. The initialization part consists of defining the required learning parameters, setting up boundaries of the search space, and generating the swarm with a random position and velocity. The search process is an iterative update of the positions and velocities. The process ends when the ending criteria are met: either the number of iterations is exceeded or the optimal solution is found. The evaluation function f and the dimensionality of vectors are problem-specific.

3.2. The behaviour of the standard PSO algorithm

Algorithm 1 contains three learning coefficients α, β, γ for adjusting the convergence abilities of the algorithm. Learning coefficients must be defined by the user according to the problem statement. No deterministic methods are available for finding their values. However, there are several non-deterministic methods for solving this problem. For example, in the case of the empirical methods we can try several parameter values and observe the behaviour of the PSO algorithm and choose the best ones. In the case of meta-heuristics, the choice of the parameter values can also be considered as a search problem, so here

Algorithm 1. Standard particle swarm optimization

```
Set bounds for the search space $B_{up}$, $B_{low}$
Set learning coefficients $\alpha$, $\beta$, $\gamma$
Define the size of the swarm $s$ and the number of iterations $e$
for $i = 0 \to s-1$ do
    Initialize particle position $p^i$ taking $B_{up}$ and $B_{low}$ into account
    $Pbest^i \leftarrow p^i$
    Initialize particle velocity $v^i$ taking $B_{up}$ and $B_{low}$ into account
    Evaluate particle $f(p^i)$
    if $f(Gbest^i) < f(p^i)$ then
        $Gbest^i \leftarrow p^i$
    end if
end for
while current iteration $< e$ and optimal solution is not found do
    for $i = 0 \to s-1$ do
        Update velocity $v^i$ for each dimension by Eq. 4
        Update position $p^i$ for each dimension by Eq. 5
        if $f(Pbest^i) < f(p^i)$ then
            $Pbest^i \leftarrow p^i$
            if $f(Gbest^i) < f(p^i)$ then
                $Gbest^i \leftarrow p^i$
            end if
        end if
    end for
end while
```



$\alpha = 0.2,\ \beta = 0.5,\ \gamma = 0.5$ $\alpha = 1.0,\ \beta = 0.5,\ \gamma = 0.5$ $\alpha = 2.0,\ \beta = 0.5,\ \gamma = 0.5$

Fig. 2. Equation: $x^2 + y^2 = 149$. Maximum number of iterations $e = 200$.

heuristic optimization can also be applied. The unknown values of the learning coefficients constitute one of the problems with the PSO algorithm.

Example 3.1. Figure 2 shows how different values of a learning coefficient can affect the optimization process. We use the equation $x^2 + y^2 = 149$ and the same initial swarm which was generated randomly. The maximum number of iterations is 200 for all three cases. The cognitive memory $\beta = 0.5$ and social memory $\gamma = 0.5$ stay the same, only inertia α is changing:

- Figure 2a shows the case where $\alpha = 0.2$. Such a small α value leads to fast convergence of the search process to the global best value *Gbest*. The exploration ability of PSO in this case is small.
- Figure 2b shows the case where $\alpha = 1.0$. The solution was found on 30 iterations. With these parameters the algorithm was able to find the optimal solution in most cases. The search process is going on inside

the 'red zone'. It means that first of all the algorithm was able to define where there is a good zone for searching (exploration) and then explore this zone closely (convergence).

- Figure 2c shows the case where $\alpha = 2.0$. In this case the optimal solution was found during 137 iterations. However, the algorithm was exploring the whole search space, so much additional work was done. We can say that in this case the algorithm does not converge to the optimal solution.

3.3. Problems with the standard PSO algorithm

As we have shown in Example 3.1, the choice of the learning coefficients for the PSO algorithm has a strong impact on optimization performance. The first problem is that no exact methods exist for defining them.

The second problem lies in the definition of Eq. 4. There is the *Gbest* position, which is valid for the whole swarm and does not take the distance between the particle and the global best position *Gbest* into account. In some cases, if *Gbest* itself is in a bad zone, the whole swarm falls into a local optima.

The second problem has two main solutions: (1) to define the *neighbourhood* of every particle by taking account of not *Gbest* for the whole swarm, but of *Gbest* for the group of 'connected' particles, (2) to define *parallel swarms*, where groups of particles move in the search space without any interactions between groups.

4. PROPOSED ALGORITHM: MODIFIED PSO BASED ON GRAVITATIONAL FIELD INTERACTIONS

To solve the problems described above, we propose not to define learning coefficients by the user, but to compute them in such a way that they take the distances between particles into account. We employ the ideas inspired from stochastic search methods based on the gravitational law.

4.1. Gravity as inspiration for optimization algorithms

Four main forces are acting in our universe: gravitational, electromagnetic, weak nuclear, and strong nuclear. These forces define the way our universe behaves and appears. The weakest force is gravitational; it defines how objects move depending on their masses.

The gravitational force between two objects *i* and *j* is directly proportional to the product of their masses and inversely proportional to the square distance between them

$$F_{ij} = G\frac{M_j \cdot M_i}{R_{ij}^2}. \tag{6}$$

Knowing the force acting on the body, we can compute acceleration as

$$a_i = \frac{F_i}{M_i}. \tag{7}$$

To construct the search algorithm based on gravity, we can use the following ideas:

- each object in the universe has mass and position;
- there are interactions between objects, which can be described by using the law of gravity;
- bigger objects (with greater mass) create a larger gravitational field and attract smaller ones.

During the last decade some researchers have tried to adapt the idea of gravity to find out optimal search algorithms. Such algorithms have some general ideas in common:

- the system is modelled by objects with mass;

- the position of the objects describes the solution and the mass of the objects depends on the evaluation function;
- the objects interact with each other using gravitational force;
- the objects with greater mass present the points in the search space with better solution.

Using these characteristics, it is possible to define the family of optimization algorithms based on gravitational force. For example, *Central Force Optimization* is a deterministic gravity-based search algorithm proposed and developed by Formato [2]. It simulates the group of probes which fly into the search space and explore it. Another algorithm, *Space Gravitational Optimization*, was developed by Hsiao et al. [3] in 2005. It simulates asteroids flying through a curved search space. A gravitationally-inspired variation of the local search algorithm, *Gravitational Emulation Local Search Algorithm*, was proposed by Webster [14] and Webster and Bernhard [15]. The newest one, the *Gravitational Search Algorithm*, was introduced by Rashedi et al. [11] as a stochastic variation of Central Force Optimization.

Basically, the gravitationally inspired algorithms are quite similar to PSO algorithms. Instead of the particle swarm we have a set of bodies with masses, ideas of the position and velocity vectors are the same, and the movement laws are similar. Our idea is to combine the two approaches to get a better one.

4.2. Existing PSO algorithm hybrids

The idea of hybridization of the PSO algorithm and gravitationally inspired search algorithms is not new. Several algorithms exist that use both ideas (PSO algorithm and gravity) to construct a heuristic search algorithm:

- PSOGSA – PSO algorithm and Gravitational Search Algorithm [7];
- extended PSO algorithm based on self-organization topology driven by fitness – PSO algorithm and Artificial Physics [8];
- Gravitational Particle Swarm [13];
- self-organizing PSO based on the Gravitation Field Model [9].

The traditional way of hybridization of the PSO algorithm with gravitationally-inspired search algorithms is to add the gravitational component to the velocity computation. Equation 4 has an additional component, which is computed by using gravitational interactions. Unfortunately, this makes the behaviour of the search algorithm even more complex and unpredictable. Additionally, the user-defined parameters still need to be found.

We propose not to add the gravitational component, but to replace the existing learning coefficients by coefficients which are computed by ‘gravitational’ interactions.

4.3. Modified PSO based on gravitational field interactions

Now we want to adapt some ideas we got from the gravitational search to the PSO algorithm. Suppose we have the current particle P, its position and velocity vectors, and its mass M, which is based on the value of the evaluation function. The position vector encodes the candidate solution, and the velocity vector presents the direction of the movement. We know the *Gbest* position and mass value M_g for this position. As for the standard PSO algorithm, so far this *Gbest* position presents the best known solution for the whole swarm. We also have the local best value for the current particle – the *Pbest* position and its mass value M_b at this point.

For now, the algorithm is similar to the standard PSO. The current particle is attracted to both *Pbest* and *Gbest* positions; in the PSO algorithm the power of this attraction and direction of the movement is controlled by learning coefficients. In our case, we recalculate the force between those three points and compute the acceleration (Fig. 3).

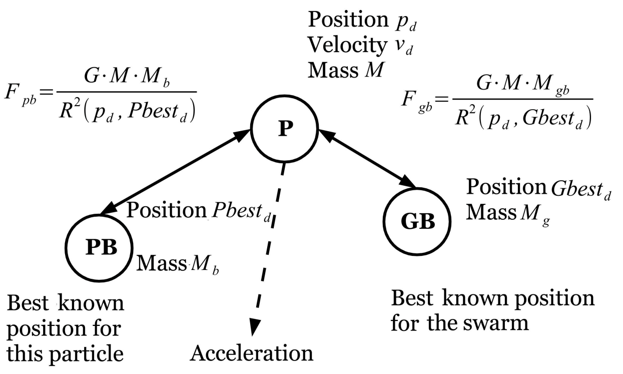

Position p_d
Velocity v_d
Mass M
$F_{pb}=\frac{G\cdot M\cdot M_b}{R^2(p_d, Pbest_d)}$
$F_{gb}=\frac{G\cdot M\cdot M_{gb}}{R^2(p_d, Gbest_d)}$
P
Position $Pbest_d$
PB
Mass M_b
GB
Position $Gbest_d$
Mass M_g
Best known position for this particle
Acceleration
Best known position for the swarm

Fig. 3. Gravitational interactions between particles.

In the proposed method we replace learning coefficients in the standard velocity computation by computed values, which are based on the idea of gravitational interactions between particles:

$$v_d(t) = \mathbf{M_i} \cdot v_d(t-1) + \mathbf{a_{pb}} \cdot r_1 \cdot (Pbest_d - p_d(t-1)) + \mathbf{a_{gb}} \cdot r_1 \cdot (Gbest_d - p_d(t-1)), \tag{8}$$

where

- M_i (instead of α) – inertial mass,
- a_{pb} (instead of β) – acceleration towards *Pbest*

$$a_{pb} = \frac{G \cdot M_b}{R^2(p(t-1), Pbest)}, \tag{9}$$

- a_{gb} (instead of γ) – acceleration towards *Gbest*

$$a_{gb} = \frac{G \cdot M_g}{R^2(p(t-1), Gbest)}, \tag{10}$$

- r_1,r_2, $Pbest_d$, $Gbest_d$, $v_d(t)$, d, $v_d(t-1)$ have the same meaning as in Eq. 4.

Now we have another movement law. Learning coefficients β and γ are replaced by the corresponding accelerations, based on gravitational interaction between the corresponding particles. The ability of the particle to save its current position is now characterized by M_i instead of α.

Contrary to the standard PSO algorithm, where learning coefficients are chosen by the end user, in our case learning coefficients are computed by using gravitational interactions between particles. Moreover, the proposed learning coefficients depend on distances between the position of the current particle *P* and *Pbest* position and the position of *P* and *Gbest* position. Thus, we have an additional property: if the current particle is far from *Gbest*, the tendency to move in that direction is small. Therefore, the behaviour of the algorithm is more stable than that of the standard PSO.

5. BEHAVIOUR OF THE PROPOSED PSO + GI METHOD

To illustrate the PSO + GI method, we perform several experiments and compare the results with other similar algorithms, such as PSO and *stochastic hill climbing (SHC)* [6].

5.1. Experimental setup

To analyse the behaviour of the proposed algorithm, we use several test problems that can be described by Eq. 2 with powers from 1 to 5. The problems are presented in Table 1. The search space is defined in the

Table 1. Test Diophantine equations

	Equation	Solutions in the range
e1	$4x+9y=91$	11
e2	$10x+7y=97$	10
e3	$24x+15y=9$	13
e4	$19x+23y=3$	4
e5	$x^2+y^2=625$	20
e6	$x^2+y^2=149$	8
e7	$x^3+y^3=1008$	2
e8	$x^4+y^4=1921$	8
e9	$x^5+y^5=19\,932$	2

range $[-50, 50]$ for each variable. The column 'Solutions in the range' shows how many solutions exist in the search space.

The initial swarm is generated in the range $[-50, 50]$ for each variable and its size is defined as 1% of all possible points in the search space. The exception is the experiment 'Bad initial set' (see Subsection 5.3). For this experiment the inital swarm is generated in the range $[-50, -45]$ for each variable, although the search space stays the same ($[-50, 50]$ for each variable).

The maximum number of iterations is taken 200. So, the algorithm stops running after 200 iterations or if the solution is found.

Two main characteristics are used for comparison: space and time. In the proposed experiments, the space is described by the number of points the algorithm checked before finding the solution and time is described by the number of iterations before the solution was reached. The number of fails (the solution was not found during 200 iterations) is an important factor as well.

For each experiment we perform 1000 runs and compute average and standard deviation for each parameter used for describing algorithm performance (see Tables 2–4).

The PSO + GI algorithm is compared to SHC and standard PSO algorithm with $\alpha = 1.0$, $\beta = 0.5$, $\gamma = 0.5$ (Section 3).

5.2. Experiments: normal workflow

Search includes two main mechanisms: global and local. The performance of each search method depends on those mechanisms. One idea of the experiments was to take a look inside and observe what exactly is going on during the search process. Usually, the search problems are multidimensional and it is hard to illustrate the search space. The use of two-dimensional DEs as test problems allows us to illustrate the behaviour of the search algorithm.

Figure 4 shows one test run of the algorithm: PSO + GI in Fig. 4a,b, SHC in Fig. 4c, and PSO in Fig. 4d for test equation e1 (Table 1).

Figure 4c illustrates the typical behaviour of SHC. There are several 'islands' that are formed by moving particles. The movements of one particle also exactly represent 'hill climbing'.

Figure 4d shows the behaviour of PSO. It is hard to define any observable pattern in the behaviour.

For PSO + GI (Fig. 4a,b) we define two main trajectories of the particles: the 'line' (Fig. 4b) and the 'orbit' (Fig. 4a). The first one appears, when *Gbest* is far from the current particle position, or constantly changing. In this case the particle moves directly to better zones. The second one, the 'orbit', appears when the particle is near *Gbest*. In this case the particle starts to move around *Gbest* in the good area. These trajectories can be explained by principles of local and global search.

Table 2 presents the results of the experiment 'Different initial set'. We performed 1000 runs for each algorithm and each test equation. A new initial set was used (generated randomly for each run). The standard PSO showed better results than the others. The PSO + GI algorithm is best for e2 and second-best for e1, e3, e5, e6, e7, and e8. Though, PSO + GI performed almost on the same level as PSO.

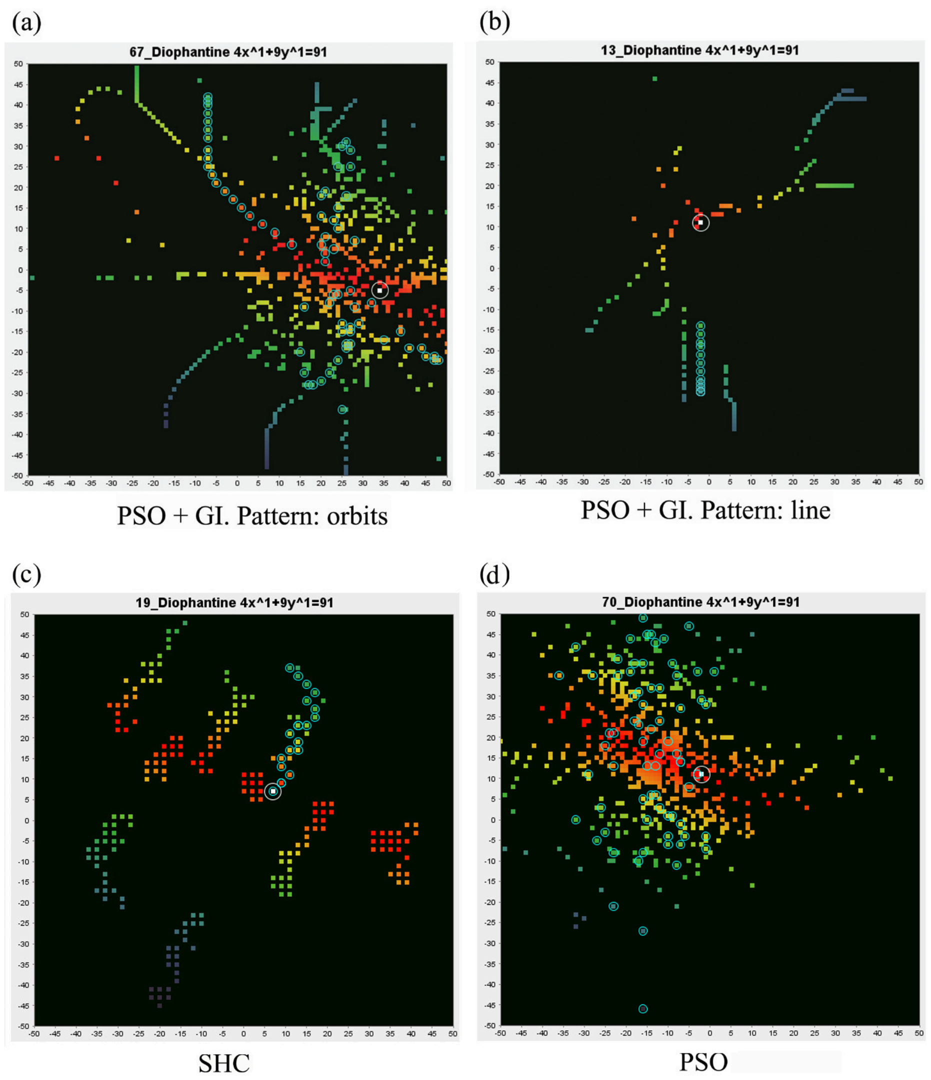


Fig. 4. Trajectories.

Table 2. Experiment I 'Different initial set'

Eq.	SHC					PSO					PSO + GI				
	Iterations		Points		Fails	Iterations		Points		Fails	Iterations		Points		Fails
	Avg.	Std. dev.	Avg.	Std. dev.		Avg.	Std. dev.	Avg.	Std. dev.		Avg.	Std. dev.	Avg.	Std. dev.	
e1	73.12	84.76	231.15	110.88	305	39.16	37.94	342.22	297.07	5	49.57	44.47	431.44	356.03	14
e2	39.06	62.79	191.93	110.13	128	49.63	49.83	412.61	366.85	30	57.22	47.72	496.78	387.39	18
e3	58.63	78.05	210.89	109.93	230	38.9	40.65	335.68	309.52	8	48.46	42.87	421.51	345.87	14
e4	66.75	80.08	247.54	112.23	261	106.08	73.20	816.15	518.88	252	111.94	70.46	919.22	541.96	270
e5	21.19	44.90	135.22	72.17	58	24.86	24.50	231.54	205.57	0	28.04	27.72	260.52	242.67	1
e6	18.69	24.29	193.67	93.84	15	34.13	36.24	307.05	291.15	6	43.04	41.48	389.31	349.68	11
e7	84.97	79.07	326.93	104.10	302	76.43	65.46	641.62	506.38	115	99.60	67.96	867.40	556.39	195
e8	19.88	22.85	214.76	101.80	13	22.33	22.68	214.41	197.49	0	32.53	31.53	305.80	277.99	3
e9	51.14	55.21	325.12	116.63	95	60.23	55.26	519.82	430.704	62	87.77	66.65	778.15	558.69	151

The behaviour of the algorithm depends not only on the mechanisms of the search, but also on the initial set. To eliminate this difference, we performed a second experiment, where we used the same initial set

Table 3. Experiment II ‘Same initial set’

Eq.	SHC					PSO					PSO + GI				
	Iterations		Points		Fails	Iterations		Points		Fails	Iterations		Points		Fails
	Avg.	Std. dev.	Avg.	Std. dev.		Avg.	Std. dev.	Avg.	Std. dev.		Avg.	Std. dev.	Avg.	Std. dev.	
e1	33.46	57.33	350.40	175.69	103	17.51	18.68	322.07	292.80	0	25.45	23.63	457.38	380.24	0
e3	13.40	33.31	205.72	153.71	29	18.97	20.13	343.78	306.74	0	23.95	23.37	425.16	375.48	0
e4	11.94	23.44	248.36	134.65	14	61.91	56.11	914.85	700.88	56	71.16	60.64	1112.82	824.05	73
e5	5.02	9.71	138.05	96.80	2	12.05	12.14	238.58	206.43	0	13.47	12.77	261.55	227.90	0
e6	3.89	1.82	125.49	44.57	0	17.16	17.32	312.60	271.15	0	22.00	19.78	411.07	334.71	0
e7	45.58	36.38	566.18	168.15	36	37.97	38.29	620.27	539.56	5	61.29	52.52	1028.17	780.42	41
e8	7.47	4.13	207.71	94.50	0	9.62	10.09	196.80	174.35	0	14.74	13.73	289.16	242.68	0
e9	9.48	5.41	253.22	114.84	0	28.37	25.69	502.13	384.52	1	53.67	46.58	929.41	718.50	22

for all 1000 runs and all three algorithms. This allows us to compare pure behaviours: how the different algorithms perform on the same initial set.

Table 3 shows the results of experiment II ‘Same initial set’. The standard PSO shows the best performance, except for the problems e4 and e9. The PSO + GI algorithm is on the second place.

5.3. Experiment: bad initial set

The results of the search depend not only on the algorithm, but also on the initial swarm. What will happen if the initial swarm is generated in the bad zone of the search space? Figure 5 shows this situation for the test equation e1. The initial swarm was generated in the range $[-50, -45]$ for each variable. For test equations, this is the area with points that have bigger evaluation function values. The PSO + GI algorithm was able to move out of the bad zone towards the better one and find the solution. However, this process takes many iterations.

All the three algorithms were able to leave the bad zone. Table 4 contains the results of the third experiment ‘Bad initial sets’. As you can see, the standard PSO performs better in most cases (except e4, e6, e7, e9; for those test problems SHC outperformed the others). The PSO + GI is the second-best algorithm for the test problem: e3 and e8.

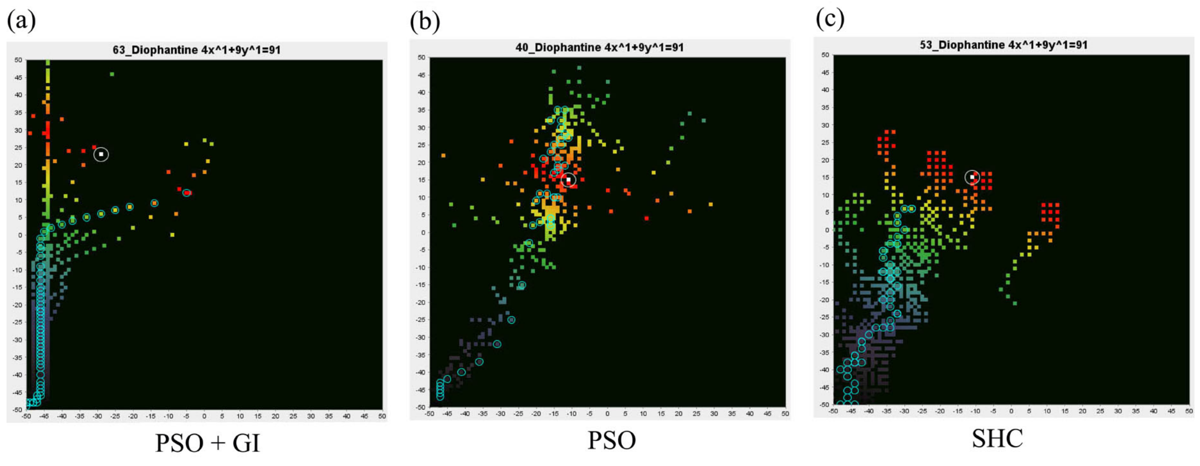


Fig. 5. Bad initial set.

Table 4. Experiment III ‘Bad initial sets’

Eq.	SHC					PSO					PSO + GI				
	Iterations		Points		Fails	Iterations		Points		Fails	Iterations		Points		Fails
	Avg.	Std. dev.	Avg.	Std. dev.		Avg.	Std. dev.	Avg.	Std. dev.		Avg.	Std. dev.	Avg.	Std. dev.	
e1	93.19	68.84	551.50	86.60	291	58.48	42.78	453.64	293.77	20	132.03	61.20	478.78	303.15	321
e2	64.42	45.17	534.92	101.38	95	70.88	50.51	544.82	349.12	52	146.65	45.55	709.76	300.31	246
e3	82.53	67.16	490.06	96.84	243	55.16	39.44	435.09	283.40	8	122.11	48.08	628.06	293.04	116
e4	86.74	68.24	512.94	106.10	262	125.39	125.39	125.39	457.02	314	164.65	47.10	945.43	354.13	515
e5	56.47	59.50	285.16	39.71	146	30.04	21.13	239.60	157.98	1	47.50	32.20	386.14	263.09	5
e6	39.53	6.42	342.87	40.55	1	36.32	29.06	294.22	214.68	6	62.25	38.81	495.14	317.34	24
e7	80.30	44.64	538.26	56.11	120	95.12	64.77	753.90	483.48	170	121.27	64.39	998.90	514.44	269
e8	45.41	17.64	374.82	44.29	12	26.26	15.46	222.10	127.88	0	44.22	34.33	400.84	299.04	7
e9	68.19	30.04	512.26	54.30	47	67.31	54.46	547.99	417.80	73	100.55	64.92	884.48	540.06	178

6. CONCLUSION AND FUTURE WORK

We proposed a modified PSO algorithm PSO + GI based on ideas of gravitational interactions between bodies. This algorithm replaces predefined learning coefficients by new ones that are calculated by means of the evaluation function and distance between particles. The general idea of the algorithm is to solve the problem with unknown learning coefficients for the standard PSO. Also, in PSO + GI, the distance factor helps to resolve the problem of the global behaviour for PSO.

The PSO + GI algorithm was tested on several Diophantine equations and the results were compared to the standard PSO and SHC. The statistics show that PSO + GI shows the second-best performance after well-tuned PSO in most cases. So the main goal of PSO + GI (to reduce the number of unknown coefficients in PSO) was achieved.

In the future, the PSO + GI algorithm should be tested for other problems with more complex search spaces.

ACKNOWLEDGEMENTS

This research was supported by Estonian Research Council institutional research grant No. IUT33-13, Estonian Doctoral School in ICT (IKTDK), and by the ERDF through the ITC project MBJSDT and Estonian national CoE project EXCS. The author is grateful to the anonymous reviewers who made constructive suggestions and remarks on the draft of this paper.

REFERENCES

1. Abraham, S., Sanya, S., and Sanglikar, M. A. Particle swarm optimization based diophantine equation solver. *CoRR*, abs/1003.2724, 2010.
2. Formato, R. A. Central force optimization: a new metaheuristic with applications in applied electromagnetics. *PIER*, 2007, **77**, 425–491.
3. Hsiao, Y.-T., Chuang, C.-L., Jiang, J.-A., and Chien, C.-C. A novel optimization algorithm: space gravitational optimization. In *Systems, Man and Cybernetics, 2005 IEEE International Conference*, Vol. 3. 2005, 2323–2328.
4. Hsiung, S. and Mattews, J. Genetic algorithm example: Diophantine equation, 1999. www.generation5.org [accessed 23 May 2015].
5. Kennedy, J. and Eberhart, R. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, Vol. 4. 1995, 1942–1948.
6. Luke, S. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available at http://cs.gmu.edu/∼sean/book/metaheuristics/ [accessed 23 May 2015].
7. Mirjalili, S. and Hashim, S. Z. M. A new hybrid PSOGSA algorithm for function optimization. In *Proceedings of International Conference on Computer and Information Application (ICCIA)*. 2010, 374–377.

8. Mo, S., Zeng, J., and Xu, W. An extended particle swarm optimization algorithm based on self-organization topology driven by fitness. *J. Comput. Inform. Syst.*, 2011, **7**(12), 4441–4454.
9. Qi, K., Lei, W., and Qidi, W. A novel self-organizing particle swarm optimization based on gravitation field model. In *American Control Conference, 2007, ACC '07*, 2007, 528–533.
10. Rashedi, E., Nezamabadi-pour, H., Saryazdi, S., and Farsangi, M. M. Allocation of static var compensator using gravitational search algorithm. In *First Joint Congress on Fuzzy and Intelligent Systems Ferdowsi University of Mashhad, Iran, August, 29–31*. 2007, 29–31.
11. Rashedi, E., Nezamabadi-pour, H., and Saryazdi, S. GSA: a gravitational search algorithm. *Inform. Sci.*, 2009, **179**(13), 2232–2248.
12. Rashedi, E., Nezamabadi-pour, H., and Saryazdi, S. Filter modeling using gravitational search algorithm. *Eng. Appl. Artif. Intell.*, 2011, **24**, 117–122.
13. Tsai, H.-C., Tyan, Y.-Y., Wu, Y.-W., and Lin, Y.-H. Gravitational particle swarm. *Appl. Math. Comput.*, 2013, **219**(17), 9106–9117.
14. Webster, B. *Solving Combinatorial Optimization Problems Using a New Algorithm Based on Gravitational Attraction*. PhD thesis, Florida Institute of Technology, Melbourne, FL, USA, 2004.
15. Webster, B. and Bernhard, P. J. *A Local Search Optimization Algorithm Based on Natural Principles of Gravitation*. Technical Report CS-2003-10, Florida Institute of Technology, 2003.
16. Zibanezhad, B., Zamanifar, K., Nematbakhsh, N., and Mardukhi, F. An approach for web services composition based on QoS and gravitational search algorithm. In *Proceedings of the 6th International Conference on Innovations in Information Technology, IIT'09*. IEEE Press, Piscataway, NJ, USA, 2009, 121–125.

Gravitatsioonilist vastasmõju arvestav osakeste parvega optimeerimise meetod

Margarita Spichakova

On esitatud osakeste parve optimeerimisalgoritmi modifikatsioon PSO + GI, kus otsimismeetodi parameetrid arvutatakse osakestevahelise gravitatsioonilise vastasmõju põhjal. PSO + GI algoritmi võib käsitleda kui hübriidi osakeste parve optimeerimise (PSO, *particle swarm optimization*) ja gravitatsioonilise optimeerimise meetodist, kus osakeste liikumistrajektoorid arvutatakse sarnaselt punktmasside liikumisvõrranditega gravitatsiooniväljas. Algoritmi tööd näidatakse kahe muutuja diofantilise võrrandi lahendamisel. Ühtlasi saab seejuures visualiseerida ja analüüsida otsimisruumi ning algoritmi käitumist kahemõõtmelisel tasandil.