10.1109/CEC.2008.4631155

Magnetic Optimization Algorithms a New Synthesis

M. H. Tayarani, N and M. R. Akbarzadeh. T.

***Abstract*—A novel optimization algorithm is proposed here that is inspired by the principles of magnetic field theory. In the proposed Magnetic Optimization Algorithm (MOA) the possible solutions are magnetic particles scattered in the search space. Each magnetic particle has a measure of mass and magnetic field according to its fitness. The fitter magnetic particles are those with higher magnetic field and higher mass. These particles are located in a lattice-like environment and apply a force of attraction to their neighbors. The proposed cellular structure allows a better exploitation of local neighborhoods before they move towards the global best, hence it increases population diversity. Experimental results on 14 numerical benchmark functions show that MOA in some benchmark functions can work better than GA and PSO.**

I. INTRODUCTION

POPULATION based optimization algorithms like Genetic Algorithms (GA), Particle Swarm Optimization (PSO) and Quantum Evolutionary Algorithms (QEA) promise more robust search due to their parallel nature but also suffer from slow convergence. The challenge in these algorithms is how to combine existing knowledge about optimization landscape to better accelerate their convergence to true optimal solutions while avoiding locally optimal solutions.

Particle Swarm Optimization is a stochastic search based on imitation of particles with a set of simple behaviors. In this algorithm the objective of all particles in the population is calculated and each particle tries to imitate the best particles in the population. Each particle finds the best particle and changes its velocity to the location of best particle. There are several works that have tried to improve the performance of the PSO algorithm. In [1] a cooperative particle swarm optimizer is proposed, where multiple swarms are used to optimize different components of solution cooperatively. A comprehensive learning particle swarm optimizer is proposed in [2], which uses a novel learning strategy based on the historical best information of all particles to update the velocity of the particles. For learning to play games, reference [3] uses PSO to train neural networks to predict the desirability of states in the leaf nodes of a game tree. To solve multimodal optimization problems and tracking multiple optima in dynamic environment, [4] proposed a species-based particle swarm optimizer. In another similar work [5] uses multiswarms PSO that are specially designed to work in dynamic environments. Reference [6] proposes a novel parameter automation strategy for the particle swarm algorithm. They proposed time-varying acceleration coefficients and inertia weight factor to control the convergence of the algorithm.

In PSO, only the best particles affect the motion of other particles and the inferior particles do not have an influence. Clearly there is some important information in the particles with low fitness that is ignored by PSO. Evolutionary Algorithms on the other hand provide better diversity by probabilistic based recombination of solutions. Coevolutionary algorithm is one approach that has emerged recently for better diversity. Reference [7] uses competitive and cooperative co-evolution to design a robust flow-path network and transporter routing for automated guided vehicles. In order to accelerate the convergence speed of the genetic algorithms, [8] proposed a novel virus co-evolutionary genetic algorithm that is based on the theory of virus evolution. In this scheme, the good genetic information can be easily transferred between individuals.

Another way to preserve diversity is by structured EAs such as, *distributed* [9] and *cellular* algorithms [10, 11]. In distributed evolutionary algorithms, the population is partitioned into a set of islands where an isolated EA is executed on each island. In Cellular EAs (cEAs) the individuals are located in a grid structure and each individual interacts with its neighbors. These types of decentralized algorithms provide a better sampling of the search space and improve the performance of EAs [10]. Rudolph and Sprave were the first to propose the cellular structure for genetic algorithm. In [12] a cellular genetic algorithm (cGA) is used for training the recurrent neural networks. In their work a local search algorithm is combined with the cellular genetic algorithms to improve the performance of GA. The hybrid of random walk algorithms and cellular genetic algorithms is used in [13] for solving the satisfiability problem. Reference [14] studies the static and dynamic cGA and analyses the exploration/exploitation tradeoff in dynamic cGA. In [15] a clustering algorithm is proposed that does not need any prior knowledge about the number of clusters. They use a Coevolutionary cGA for optimizing the model.

This paper proposes a new optimization algorithm based on the principles of the attraction among magnetic particles. In this algorithm the particles are attracted to their neighbors in a lattice-like structured population based on their magnetic field. Each particle in the search space, even the worst of them, applies an attractive force to the other neighboring particles. This characteristic makes the algorithm capable effectively to search the optimization landscape as will be shown in this paper. The value of this magnetic force is related to the magnetic field of each particle and the distance between them. Experimental results on 14 benchmark functions show that in most of the problems the proposed algorithm has a better performance than conventional GA and PSO.

This paper is organized as follows. Section II introduces

M. H. Tayarani. N (IEEE Student Member) is with the Department of Computer Engineering, Azad University. (e-mail: tayarani@ieee.org).

M.- R. Akbarzadeh-. T (IEEE Senior Member) is with the Department of Electrical Engineering, Ferdowsi University of Mashhad. (e-mail: akbarzadeh@ieee.org)

the Magnetic Optimization Algorithms. In Section III the best parameters for each of the 14 numerical benchmark functions are separately determined. Section IV analyses the size of population and its effect on PSO, GA, and MOA performance. And finally Section V tests the proposed algorithm on 14 numerical functions and compares it with PSO and GA.

II. MAGNETIC OPTIMIZATION ALGORITHMS

There are four types of forces in our universe and electromagnetic force is one of them. This kind of force has long-range effect; meaning its effect decreases as distance between two particles increases. This type of force disappears only when the distance of two particles is infinity. In the proposed algorithm the possible solutions are each a magnetic particle with attractive long-range force.

In MOA the magnetic particles operate in a lattice like interaction environment as shown in Fig. 1.

The pseudo code of the proposed optimization algorithm is briefly shown below and is described in the following steps.

```
Procedure MOA
begin
       t=0
   1.  initialize X^0 with a lattice-like
       structure.
   2.  while not termination condition do
       begin
          t=t+1
   3.     evaluate the individuals in X^t and
             store their profits in magnetic
             field B^t
   4.     normalize B^t according to (1)
   5.     evaluate the mass M^t for all
             particles according to (2)
   6.     for all particles x^t_ij in X^t do
           begin
   7.        F_ij=0
   8.        find N_ij
   9.        for all x^t_uv in N_ij do
              begin
  10.           F_ij,k = F_ij,k + (x^t_uv,k - x^t_ij,k) x B^t_uv / D(x^t_ij,k, x^t_uv,k)
              end
           end
  11.     for all particles x^t_ij in X^t do
           begin
  12.        v^{t+1}_ij,k = (F_ij,k / M_ij,k) x R(l_k, u_k)
  13.        x^{t+1}_ij,k = x^t_ij,k + v^{t+1}_ij,k
           end
       end
end
```

The pseudo code of MOA is described as below:

1. In this step all of the particles in the population X^t (for t=0) are initialized randomly. The initialization step is performed as:

   $x^t_{ij,k} = \mathrm{R}(l_k, u_k)$ for i,j=1,2,…,S, k=1,2,…,m and t=0

   Where i,j indicates the location of the particle in the lattice, t is the iteration number of MOA, S is the size of lattice-like environment, m is the dimension of the search space, l_k and u_k are the lower bound and the upper bound of the k-th dimension of search space respectively, and R(.,.) is a uniform random number generator.

2. The while loop is terminated when the termination condition is satisfied. Termination condition here is when maximum number of iterations is reached.
3. In this step the objective of each particles x^t_{ij} in X^t is calculated and stored in the magnetic field, B^t_{ij}. Where t is the iteration number and i,j (i=1,2,…,S) shows the location of the particle in the population.
4. Next the normalization is performed on the B^t. The normalization is performed as:

   $$B_{ij} = \frac{B_{ij} - Min}{Max - Min} \quad (1)$$

   Where:

   $$Min = \underset{i,j=1}{\overset{S}{\text{minimum}}}(B^t_{ij})$$

   $$Max = \underset{i,j=1}{\overset{S}{\text{maximum}}}(B^t_{ij})$$

5. In this step the mass of all particles is calculated and stored in M^t:

   $$M^t_{ij} = \alpha + \rho \times B^t_{ij} \quad (2)$$

   Where α and ρ are constant values.
6. In this step in the "for" loop, the resultant force of all forces on each particle is calculated.
7. At first the resultant force which is applied to particle x^t_{ij} (F_{ij}) is set to zero.
8. In the lattice-like structure of MOA, each particle interacts only with its neighbors i.e. each particle applies its force only to its neighbors. In this step the neighbors of x^t_{ij} is found. The set of neighbors for particle x_{ij} is defined as:

   $N_{ij} = \{x_{i'j}, x_{ij'}, x_{i''j}, x_{ij''}\}$

   Where:

   $$i' = \begin{cases} i-1 & i \neq 1 \\ S & i = 1 \end{cases}, \qquad j' = \begin{cases} j-1 & j \neq 1 \\ S & j = 1 \end{cases}$$

   $$i'' = \begin{cases} i+1 & i \neq S \\ 1 & i = S \end{cases}, \quad j'' = \begin{cases} j+1 & j \neq S \\ 1 & j = S \end{cases}$$

9. In this step the force which is applied to particle x^t_{ij} from its neighbor's x^t_{uv} ($\forall x^t_{uv} \in N_{ij}$) is calculated.
10. The force which is applied from x^t_{uv} to x^t_{ij} is related to the distance between two particles and is calculated

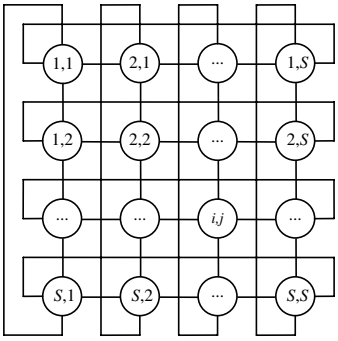

Fig.1. The proposed cellular lattice with the size of S

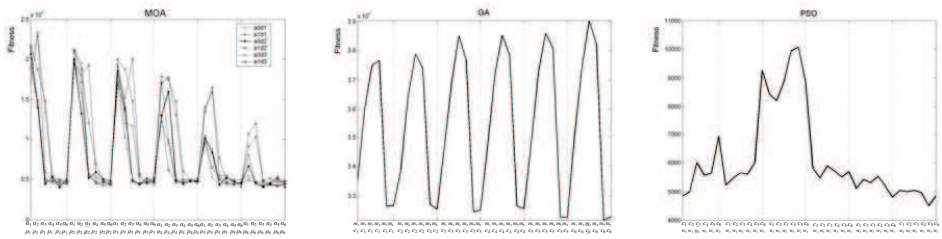

Fig.2. Parameter setting for MOA, GA, PSO for Generalized Rastrigin' s Function. The parameters of MOA is set to $\alpha_1 \ldots \alpha_6$=(0.1,1,4,8,15,30), $\rho_1 \ldots \rho_6$= (0.1,1,4,8,15,30). a_0 means using equation (7) and a_1 means using equation (8) for updating the location of particles and d_1 means using distance$_1$ in equation (3). The mutation rate and crossover rate of GA is $m_1 \ldots m_6$=(0.001,0.002,0.004,0.008,0.016,0.02), $c_1 \ldots c_6$=(0.1,0.2,0.4,0.6,0.8,1). The parameters of PSO is set at $c_1 \ldots c_6$=(0.002,0.005,0.01,0.05,0.1,0.4) and $w_1 \ldots w_6$=(0.2,0.5,1,1.2,1.5,2).

as:

$$F_{ij,k} = \frac{(x^t_{uv,k} - x^t_{ij,k}) \times B^t_{uv}}{D(x^t_{ij,k}, x^t_{uv,k})} \quad (3)$$

Where $D(.,.)$ is the distance of two particles and calculated as:

$$D_1(x^t_{ij}, x^t_{uv}) = \frac{1}{m}\sum_{k=1}^{m}\left|\frac{x^t_{ij,k} - x^t_{uv,k}}{u_k - l_k}\right| \quad (4)$$

or

$$D_2(x^t_{ij}, x^t_{uv}) = \sqrt{\frac{1}{m}\sum_{k=1}^{m}\left(\frac{x^t_{ij,k} - x^t_{uv,k}}{u_k - l_k}\right)^2} \quad (5)$$

or

$$D_3(x^t_{ij}, x^t_{uv}) = \max_{k=1}^{m}\left|\frac{x^t_{ij,k} - x^t_{uv,k}}{u_k - l_k}\right| \quad (6)$$

Where u_k and l_k are the upper and lower bounds of the k-th dimension of the search space respectively, x^t_{ij} and x^t_{uv} are i,j-th and u,v-th particles of the population in iteration t respectively and $x^t_{ij,k}$ is the k-th dimension of i,j-th particle in iteration t.

The distance between each two particles is normalized here between [0,1] by considering the spread of the search space. Otherwise, the applied force to each particle in the problems with smaller domain of search space would be higher than the problems with larger domain of search space. For instance, the convergence speed in problems with small search space, say S=[-1,-1]^m would be faster than the problems with large search space, say S=[-100,-100].

11. In the "for" loop, the movement for all particles is calculated as in (7) or (8).
12. The velocity and the movement of each particle is calculated as:

$$v^{t+1}_{ij,k} = \frac{F_{ij,k}}{M_{ij,k}} \times \mathrm{R}(l_k, u_k)$$

$$x^{t+1}_{ij,k} = x^t_{ij,k} + v^{t+1}_{ij,k} \quad (7)$$

Additionally we can also consider acceleration for particles as follows:

$$a^{t+1}_{ij,k} = \frac{F_{ij,k}}{M_{ij,k}} \times \mathrm{R}(l_k, u_k)$$

$$v^{t+1}_{ij,k} = v^t_{ij,k} + a^{t+1}_{ij,k}$$

$$x^{t+1}_{ij,k} = x^t_{ij,k} + v^{t+1}_{ij,k} \quad (8)$$

III. FINDING THE BEST PARAMETERS

For evaluating the proposed algorithm, MOA is compared with PSO and GA. In MOA, GA and PSO there are some parameters that must be found. For comparing these algorithms at first the best parameters for these algorithms are found. This Section tries to find the best parameters α and ρ in (2) and the best distance measure, (4), (5) or (6) for MOA. In MOA we can use (7) or (8) for updating the location of the particles. This paper also tries to find the better updating operator for the location of the particles. This treatment for finding optimal parameters is similar to [20] where a systematic way is illustrated to ensure a fair and true comparison of optimization algorithms on their best set of parameters.

In this paper the PSO algorithm is considered as [16]:

$$v^{t+1}_i = wv^t_i + c_1\,\mathrm{R}(0,1)(\mathrm{pbest}^t_i - x^t_i) + c_2\,\mathrm{R}(0,1)(\mathrm{gbest}^t - x^t_i) \quad (9)$$

$$x^{t+1}_i = x^t_i + v^{t+1}_i$$

Where c_1 and c_2 are two positive constants, R(.,.) is a uniform random number generator, w is the inertia weight, pbest^t_i is the best position which the particle i has achieved, gbest^t is the best position which the overall swarm has achieved and t is the iteration number of the algorithm.

There are some parameters in GA and PSO that must be tuned (crossover rate and mutation rate in GA and c_1, c_2 and w in PSO). In PSO we considered c_1= c_2. For finding these parameters we used 14 numerical benchmark functions (see appendix A). It is obvious that the best parameters for each algorithm are problem dependent, so we find the best parameters for each benchmark function independently.

The mutation in GA is considered as: for each allele of each chromosome, mutate the allele with probability of mutation. The changing operation is performed as changing the value of allele to a random number between the lower bound and the upper bound of the search space. The pseudo code of mutation operator is considered as:

```
Procedure mutation
 begin
 1.  for all individuals x^t_i in X^t do
 2.    for all alleles x^t_{i,k} in x^t_i do
 3.      if R(0,1)<mutation rate
 4.        x^t_{i,k}=R(l_k,u_k)
 end
```

Where R(.,.) is a uniform random number generator, $x^t_{i,k}$ is k-th allele of i-th individual in the population in generation t and l_k , u_k are the lower bound and the upper bound of the search space respectively.

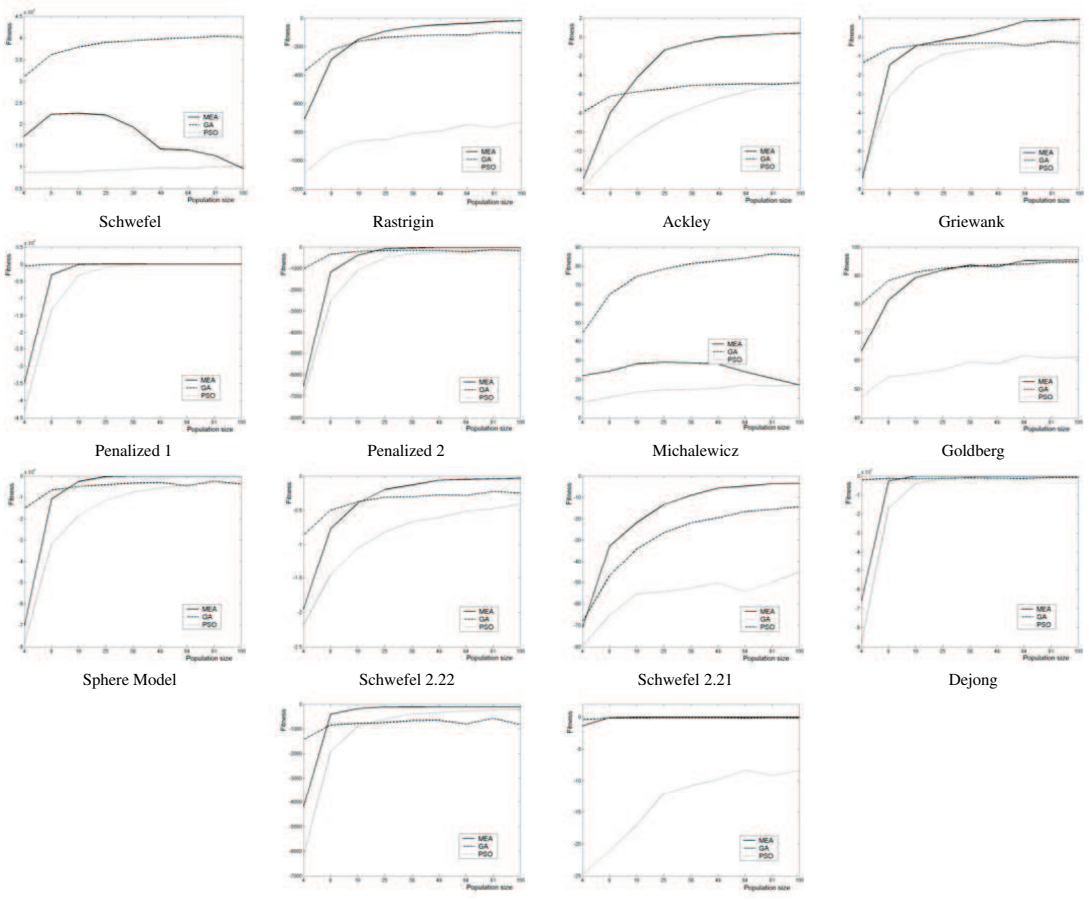

Fig 3. The effect of population size on the performance of MOA, PSO and GA. The results are the average of the best solution found over 50 runs.

Fig. 2 shows the parameter setting of MOA, GA and PSO for Generalized Rastrigin' s Function. In this figure the horizontal axis is the parameters of the algorithm and the vertical axis is the best fitness averaged over 50 runs. Table I summarizes the best parameters for MOA, GA and PSO for the 14 benchmark functions. The method of finding the best parameters for the algorithms for all benchmark functions is the same as in Fig 2.

TABLE I
PARAMETER SETTING FOR 14 NUMERICAL BENCHMARK FUNCTIONS. $a = 0$ MEANS USING (7) AND $a = 1$ MEANS USING (8) FOR UPDATING THE LOCATION OF PARTICLES AND $D = 1$ MEANS USING D_1 IN (3)

	MOA				PSO		GA	
	a	D	α	ρ	c_1, c_2	w	m	c
Schwefel	1	1	1	0.1	0.1	1	0.004	1
Rastrigin	1	2	4	1	0.01	0.2	0.004	1
Ackley	0	2	1	0.1	0.05	0.2	0.008	1
Griewank	0	3	1	0.1	0.01	0.2	0.004	1
Penalized 1	1	3	1	1	0.005	0.5	0.004	1
Penalized 2	0	3	0.1	0.1	0.01	0.2	0.004	1
Michalewicz	1	1	4	1	0.4	0.5	0.004	1
Goldberg	1	3	1	0.1	0.005	0.5	0.004	1
Sphere Model	1	3	1	0.1	0.01	0.2	0.004	1
Schwefel 2.22	0	3	1	0.1	0.005	0.5	0.004	1
Schwefel 2.21	0	3	0.1	0.1	0.01	0.5	0.008	0.8
Dejong	1	2	1	1	0.005	0.5	0.008	0.1
Rosenbrock	0	2	1	0.1	0.005	0.2	0.004	1
Kennedy	1	3	0.1	0.1	0.005	1	0.008	0.1

This section finds the best parameters for each problem independently. This is because we try to compare the algorithms in their best conditions and does not mean it is necessary to find the best parameters for each problem before the process of optimization. On the other hand the parameters of the algorithm for the different problems is not much variable and the effect of parameters on the performance of the algorithm is not considerable. This is observable in Table I, which the values of the parameters for various problems are similar. So for a new problem we can use any parameters of the Table I which is better for most problems.

IV. SIZE OF POPULATION

In Evolutionary Algorithms the size of population is one of the main parameters that affect the robustness and computational efficiency of the algorithm [17]. Naturally increasing the size of the population increases the effectiveness of EAs. This is because of two reasons. Firstly increasing the size of population increases the number of particles which are searching the search space. Each particle

TABLE II

EXPERIMENTAL RESULTS OF THE FOURTEEN NUMERICAL FUNCTION OPTIMIZATION PROBLEMS. THE NUMBER OF RUNS WAS 50. **MEAN** AND **STD** REPRESENT THE MEAN OF BEST ANSWER AND STANDARD DEVIATION OF BEST ANSWERS FOR 50 RUNS RESPECTIVELY. m IS THE DIMENSION OF PROBLEM.

	$m=100$						$m=250$					
	MOA		GA		PSO		MOA		GA		PSO	
	MEAN	**STD**	**MEAN**	**STD**	**MEAN**	**STD**	**MEAN**	**STD**	**MEAN**	**STD**	**MEAN**	**STD**
Schwefel	2.2×10^{4}	2.7×10^{3}	3.9×10^{4}	4.1×10^{2}	9.5×10^{3}	1.7×10^{3}	4.6×10^{4}	5.8×10^{3}	8.5×10^{4}	1.4×10^{3}	1.5×10^{4}	2.1×10^{3}
Rastrigin	-82.8	20.4	-136.5	12.9	-830.1	46.7	-5.7×10^{2}	6.57×10^{1}	-8.4×10^{2}	4.5×10^{1}	-2.3×10^{3}	7.8×10^{1}
Ackley	-1.3	0.63	-5.2	0.4	-8.5	0.6	-5.2	0.4	-12.5	0.24	-9.1	0.38
Griewank	-0.15	0.05	-0.37	0.56	-1.03	0.15	-1.6	0.2	-3.9	0.3	-2.7	0.26
Penalized 1	60.95	5.6	42.55	8.5	-508.1	442.6	-1.5×10^{2}	3.1×10^{2}	-2.5×10^{4}	3.6×10^{3}	-2.7×10^{3}	1.1×10^{3}
Penalized 2	-78.5	43.0	-184.6	31.8	-533.4	132.5	-7.2×10^{2}	1.2×10^{2}	-2.8×10^{3}	2.8×10^{2}	-1.6×10^{3}	2.4×10^{2}
Michalewicz	29.0	2.7	78.3	1.5	14.2	2.1	4.5×10^{1}	4.9	1.5×10^{2}	3.9	2.2×10^{1}	3.5
Goldberg	92.9	1.6	92.5	0.59	57.6	2.2	2.1×10^{2}	4.3	2.1×10^{2}	1.9	1.3×10^{2}	3.7
Sphere Model	-3.7×10^{2}	2.6×10^{2}	-3.8×10^{3}	5.9×10^{2}	-1.14×10^{4}	1.9×10^{3}	-1.06×10^{4}	1.9×10^{3}	-4.3×10^{4}	-2.9×10^{3}	-3.1×10^{4}	2.7×10^{3}
Schwefel 2.22	-0.19	0.04	-0.32	0.02	-0.82	0.07	-0.56	0.03	-0.81	0.04	-0.88	0.05
Schwefel 2.21	-13.88	2.67	-53.4	3.3	-26.7	2.8	-24.9	1.87	-76.8	2.09	-30.6	2.3
Dejong	-1.4×10^{2}	1.4×10^{2}	-1.1×10^{4}	3.7×10^{3}	-1.5×10^{4}	4.9×10^{3}	-1.5×10^{4}	5.1×10^{3}	-1.14×10^{6}	1.68×10^{5}	-1.2×10^{5}	2.37×10^{4}
Rosenbrock	-1.1×10^{2}	7.6	-7.2×10^{2}	9.2×10^{1}	-5.6×10^{2}	9.1×10^{1}	-5.7×10^{2}	5.7×10^{1}	-4.1×10^{3}	3.9×10^{2}	-1.6×10^{3}	1.6×10^{2}
Kennedy	-3.3×10^{-2}	1.6×10^{-4}	-1.3×10^{-1}	1.7×10^{2}	-1.4×10^{1}	3.1	-9.3	1.2	-1.3	0.14	-61.9	3.4

	$m=500$						$m=1000$					
	MOA		GA		PSO		MOA		GA		PSO	
	MEAN	**STD**	**MEAN**	**STD**	**MEAN**	**STD**	**MEAN**	**STD**	**MEAN**	**STD**	**MEAN**	**STD**
Schwefel	7.4×10^{4}	8.8×10^{3}	1.3×10^{5}	1.9×10^{3}	2.09×10^{4}	3.6×10^{3}	1.16×10^{5}	1.59×10^{4}	1.93×10^{5}	3.12×10^{3}	2.96×10^{4}	4.26×10^{3}
Rastrigin	-2.1×10^{3}	1.9×10^{2}	-3.05×10^{3}	8.5×10^{1}	-4.7×10^{3}	1.4×10^{2}	-6.0×10^{3}	2.27×10^{2}	-8.9×10^{3}	1.34×10^{2}	-9.78×10^{3}	1.6×10^{2}
Ackley	-7.27	0.31	-15.4	0.12	-9.2	0.21	-8.5	0.17	-16.8	0.079	-9.3	0.16
Griewank	-4.7	0.3	-23.05	0.89	-5.7	0.35	-11.02	0.54	-87.3	2.47	-11.6	0.4
Penalized 1	-2.3×10^{3}	1.02×10^{3}	-1.7×10^{5}	8.4×10^{3}	-7.4×10^{3}	1.98×10^{3}	-1.14×10^{4}	2.3×10^{3}	-6.14×10^{5}	1.59×10^{4}	-1.7×10^{4}	2.77×10^{3}
Penalized 2	-2.4×10^{3}	3.24×10^{2}	-2.1×10^{4}	1.16×10^{3}	-3.6×10^{3}	3.5×10^{2}	-6.59×10^{3}	5.9×10^{2}	-9.0×10^{4}	2.8×10^{3}	-7.7×10^{3}	5.0×10^{2}
Michalewicz	6.2×10^{1}	8.1	2.2×10^{2}	5.05	3.17×10^{2}	4.7	8.4×10	9.1	3.0×10^{2}	6.3	4.4×10^{1}	6.9
Goldberg	3.7×10^{2}	7.5	3.5×10^{2}	3.1	2.6×10^{2}	4.5	6.5×10^{2}	8.5	5.9×10^{2}	5.1	5.0×10^{2}	7.7
Sphere Model	-4.1×10^{4}	3.9×10^{3}	-2.4×10^{5}	1.0×10^{4}	-6.3×10^{4}	4.8×10^{3}	-1.1×10^{5}	7.7×10^{3}	-9.4×10^{5}	2.5×10^{4}	-1.3×10^{5}	6.2×10^{3}
Schwefel 2.22	-7.6×10^{-1}	3.5×10^{2}	-1.5	5.3×10^{-2}	-8.9×10^{-1}	3.0×10^{-2}	-0.86	0.02	-2.25	0.03	-0.91	0.02
Schwefel 2.21	-2.9×10^{1}	1.7	-8.7×10^{1}	1.3	-3.37×10^{1}	2.8	-33.4	1.77	-93.4	0.73	-36.1	2.4
Dejong	-2.1×10^{5}	5.5×10^{4}	-1.9×10^{7}	1.6×10^{6}	-5.4×10^{5}	8.6×10^{4}	-1.67×10^{6}	2.05×10^{5}	-1.92×10^{8}	9.18×10^{6}	-2.4×10^{6}	2.13×10^{5}
Rosenbrock	-1.9×10^{3}	1.39×10^{2}	-1.9×10^{4}	1.1×10^{3}	-3.4×10^{3}	2.3×10^{2}	-5.55×10^{3}	2.34×10^{2}	-8.04×10^{4}	2.83×10^{3}	-7.02×10^{3}	2.93×10^{2}
Kennedy	-5.1×10^{1}	2.57	-1.3×10^{1}	9.9×10^{-1}	-1.3×10^{2}	3.7	-1.61×10^{2}	3.87	-76.58	2.89	-2.73×10^{2}	5.52

searches the search space and tries to find better solutions. So an evolutionary algorithm with higher numbers of possible solutions has a higher searching ability. Secondly this is because of the swarm intelligence emerges as a result of the cooperation, coordination and competition among agents. For example in GA, the crossover between two inferior individuals can sometimes make a superior offspring. Or crossing over an inferior individual with a superior individual makes the algorithm able to escape a local optimum and find the global optimum. Increasing the size of the population promotes more interaction among the individuals and thereby more swarm intelligence. This Section discusses the effects of population size on the performance of the proposed algorithm and two main algorithms PSO and GA. Fig. 3 shows the effects of population size on the performance of PSO, GA and MOA. The size of problem is set to m=100 and the parameters of the algorithms (crossover rate and mutation rate in GA, c_1 ,c_2 and w in PSO and α, β, distance measure and update operator in MOA) are set at the best values which are found in Section III. As seen, the size of the population can improve the performance of these algorithms. But the rate of improving the performance in MOA is higher than PSO and GA. There are two reasons for effect of population size on the performance of EAs. The second reason that we mentioned does not appear in PSO and GA, so increasing of the population size has a smaller effect on these algorithms. In PSO each particle tries to change its velocity to the best position which it has achieved and the best position which the overall swarm has achieved. It means only the best particle can affect the other particles and the inferior particles can not affect each other. On the other hand in MOA, all of the possible solutions affect each other. In MOA even the inferior possible solutions affect the other particles. In MOA the inferior magnetic particles attract the other particles. This interaction among all of the possible solutions improves the swarm intelligence of MOA, the characteristic which does not appear in PSO. Increasing the size of the population can not improve the performance of PSO well because in this algorithm only the best possible solutions guide the algorithm to the optima and increasing the number of ineffective particles can not improve the swarm intelligence. The small effect of the population size on PSO and GA is because of the local search of new particles. On the other hand in MOA increasing the population size makes new particles for local search and also improves the interaction between the particles that improves the swarm intelligence of the algorithms.

The main idea of the PSO and MOA seems to be similar, but the main difference of these algorithms is the interaction of the particles. The interaction between the particles in MOA is higher than PSO so MOA has a better swarm intelligence than PSO. There are only two exceptions in the considered examples. Increasing the size of population only

in Generalized Schwefel's function 2.26 and Michalewicz Function does not improve the performance of MOA. This is because MOA does not work well on these two functions (see experimental results).

V. EXPRIMENTAL RESULTS

The population size of MOA for all of the experiments is set to 25 (S=5) and for PSO and GA is 25, and maximum iteration termination condition is used as maximum generation equal to 1000. All results were averaged over 50 runs. The parameters of MOA, GA and PSO are set to the best ones which are found in Section III.

Table II summarizes the experimental results of MOA, GA and PSO for m=100, 250, 500 and 1000 for 14 benchmark functions. As indicated by all of the benchmark functions, proposed algorithm consistently has a better performance than PSO. The proposed algorithm is better than GA in 12 of the benchmark functions whereas the performance of GA in Generalized Schwefel's function 2.26 and Michalewicz Function is better than MOA.

VI. CONCLUSION

This paper proposes a novel optimization algorithm called Magnetic Optimization Algorithms. The cellular structure of the MOA has two advantages. Firstly the proposed cellular structure maintains a better diversity for the population, secondly the applied force between all of the particles is not calculated and only the force between the neighbors is calculated so computation is not exceeds. This type of connectivity can decrease the computational complexity. Calculating the force between all of the particles in a population with n particles has a time complexity of $O(n^2)$ on the other hand the time complexity of calculating the force in cellular structure is $O(n)$. This paper also tries to find the best parameters for the proposed algorithm.

REFERENCES

[1] Frans van den Bergh and Andries P. Engelbrecht,"A Cooperative Approach to Particle Swarm Optimization," *IEEE Trans. Evol. Comput*. vol. 8. No 3, pp. 225–239, 2004.

[2] J. J. Liang, A. K. Qin, Ponnuthurai Nagaratnam Suganthan, and S. Baskar, "Comprehensive Learning Particle Swarm Optimizer for Global Optimization of Multimodal Functions" *IEEE Trans. Evol. Comput*. vol. 10. No 3, pp. 281–295, 2006.

[3] Leon Messerschmidt and Andries P. Engelbrecht, "Learning to Play Games Using a PSO-Based Competitive Learning Approach" *IEEE Trans. Evol. Comput*. vol. 8. No 3, pp. 280–288, 2004.

[4] Daniel Parrott, Xiaodong Li, "Locating and Tracking Multiple Dynamic Optima by a Particle Swarm Model Using Speciation" *IEEE Trans. Evol. Comput*. vol. 10. No 4, pp. 440–458, 2006.

[5] Tim Blackwell, Jürgen Branke, "Multiswarms, Exclusion, and Anti-Convergence in Dynamic Environments" *IEEE Trans. Evol. Comput*. vol. 10. No 4, pp. 459–472, 2006.

[6] Asanga Ratnaweera, Saman K. Halgamuge, Harry C. Watson, "Self-Organizing Hierarchical Particle Swarm Optimizer with Time-Varying Acceleration Coefficients" *IEEE Trans. Evol. Comput*. vol. 8. No 3, pp. 240–255, 2004.

[7] Ryosuke Chiba, Jun Ota and Tamio Arai, "AGV System Design using Competitive and Cooperative Co-evolution," *SICE-ICASE International Joint Conference* 2006 Oct. 18-21, 2006 in Bexco, Busan, Korea.

[8] Han Zhao, Xiansheng Gao and Lingyun Zhu, "A Virus Co-evolution Genetic Algorithm Based on Niche Technology," *Proceedings of the* 2006 *IEEE International Conference on Information Acquisition August* 20 - 23, 2006, Weihai, Shandong, China.

[9] E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms*, 2nd Ed, ser. Book Series on Genetic Algorithms and Evolutionary Computation. Norwell, MA: Kluwer, 2000, vol. I.

[10] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Trans. Evol. Comput*. vol. 6, no. 5, pp. 443–462, Oct. 2002.

[11] Rudolph, G. Sprave, J "A cellular genetic algorithm with self-adjusting acceptance threshold," *IEEE First International Conference on Genetic Algorithms in Engineering Systems*, 1995.

[12] Ku, K.W.C. Man Wai Mak Wan Chi Siu, "Adding learning to cellular genetic algorithms for training recurrent neural networks," *IEEE Trans. Neural Networks*. vol. 10. No 2, pp. 239–252, 1999.

[13] Folino, G. Pizzuti, C. Spezzano, G. "Parallel hybrid method for SAT that couples genetic algorithm local search," *IEEE Trans. Evol. Comput*. vol. 5. No 4, pp. 323–334, 2001.

[14] Alba, E. Dorronsoro, B "The exploration/exploitation tradeoff in dynamic cellular genetic algorithms," *IEEE Trans. Evol. Comput*. vol. 9. No 2, pp. 126–142, 2005.

[15] C.J. Veenman, M.J.T. Reinders, and E. Backer, "A Cellular Coevolutionary Algorithm for Image Segmentation," *IEEE Trans. Image Processing*. vol. 12. No 3, pp. 304–316, 2005.

[16] M. R Alrashidi and M. E. El-Hawary "A Survey of Particle Swarm Optimization Applications in Electric Power Systems" *IEEE Trans. Evol. Comput*. vol. 9, pp. 1–6, 2005.

[17] V. K. Koumousis, C. P. Katsaras. "A Saw-Tooth Genetic Algorithm Combining the Effects of Variable Population Size and Reinitialization to Enhance Performance," IEEE Trans. Evol. Comput. vol. 10, pp. 19–28, 2006.

[18] W. Zhong, J. Liu, M. Xue, L. Jiao, "A Multi-agent Genetic Algorithm for Global Numerical Optimization," IEEE Trans. Sys, Man and Cyber. vol. 34, pp. 1128–1141, 2004.

[19] A.-R. Khorsand, M.-R. Akbarzadeh-T. "Quantum Gate Optimization in a Meta-Level Genetic Quantum Algorithm". IEEE International Conference on Systems, Man and Cybernetics, 2005.

[20] K. Han, J. Kim, "Quantum-inspired evolutionary algorithm for a class of combinatorial optimization," *IEEE Trans. on Evolutionary Computation*, vol. 6, no. 6, 2002.

APPENDIX A

Global numerical optimization problems arise in many fields of science, engineering, and business. Since many of these problems cannot be solved analytically, GAs becomes one of the popular methods to address them [18]. There are some benchmark numerical functions for testing the optimization algorithms. Here we used 14 benchmark functions for testing the algorithms:

Generalized Schwefel's Problem 2.26 [18], Generalized Rastrigin's Function [18], Ackley's function [18], Generalized Griewank Function [18], Generalized Penalized Function 1 [18], Generalized Penalized Function 2 [18], Michalewicz Function [19], Goldberg & Richardson Function [17], Sphere Model [18], Schwefel's Problem 2.22 [18], Schwefel's Problem 2.21 [18], Dejong Function 4 [19], Rosenbrock Function [17], and Kennedy multimodal function generator [17].

These functions have several local minima and one global minimum. Since they are used for maximization process, we use $-f(x)$ as fitness function.