MAGNETIC PARTICLE SWARM OPTIMIZATION

Paulo S. Prampero¹ and Romis Attux²

¹Federal Institute of Science, Technology and Education of So Paulo – Salto, Brazil
Department of Computer Engineering and Industrial Automation
FEEC - University of Campinas (Unicamp) Campinas, Brazil

²Department of Computer Engineering and Industrial Automation
FEEC – University of Campinas (Unicamp) Campinas, Brazil

Abstract

This paper presents and analyzes a search paradigm called Magnetic Particle Swarm Optimization. This paradigm gives support to two algorithms that combine elements of the behavior of magnetic dipoles within a framework that includes several elements that are known to be essential to effective multimodal search. The algorithms are applied to a variety of functions and their performance is compared with those of a number of related well-established metaheuristics. In addition to that, convergence and sensitivity analyses are presented for the first time.

1 Introduction

In this work, we present a detailed analysis of the search paradigm called Magnetic Particle Swarm Optimization (MPSO) [1][2]. The paradigm is described in terms of its two representatives, the standard MPSO algorithm [1] and the variant called MPSO with estimation of distribution [2].

The raison d’être of the MPSO approach is twofold: 1) to build search tools inspired in elements of the behavior of magnetic dipoles and 2) to define an operation structure in accordance with features of well-established global optimizers. Therefore, the proposal gravitates around the perspective of efficiently performing multimodal search.

In this work, a detailed comparison between the algorithms is carried out for a representative set of benchmark functions and related metaheuristics (including classical PSO, an electromagnetic strategy and differential-evolution based techniques). It is important to stress that the presented results include, for the first time, elements of convergence and sensitivity analyses.

The paper is structured as follows. In Section 2, the classical particle swarm optimization approach is presented, together with an overview of pertinent extensions. Section 3 presents a metaheuristic inspired in electromagnetic theory, more specifically on the behavior of electrical charges. Section 4 brings the formulation known as differential evolution, while section 5 discusses the idea of estimation of distribution and section 6 presents a combination between this idea and differential evolution. In section 7, the MPSO paradigm is presented and explained in detail. Section 8 brings the results and section 9 contains the conclusions and perspectives for future work.

2 Particle Swarm Optimization

The Particle Swarm Optimization (PSO) algorithm is, essentially, a search methodology inspired in the notion of social adaptation of knowledge. The technique was developed by Kennedy and Eberhart in 1995 [3], and, since then, has received a great deal of attention of the natural computing scientific community.
In the context of PSO, each solution to the problem of interest corresponds to a particle, and each particle is capable of changing its position in the search space. The way this movement is performed is typically determined by a combination of two terms, one that corresponds to individual knowledge and one that models the cognitive effect of social interaction. The cost function to be optimized is used to evaluate the degree of adequacy of a particle.

In simple terms, a PSO algorithm can be understood in terms of three main stages [4]:

- **Evaluate**: the particles must have the ability to sense the environment and to evaluate their own behavior.
- **Compare**: a particle must use others as references for comparison.
- **Imitate**: successful behavior of neighbors tends to influence the history of a particle.

The rationale of these three stages and the manner whereby they are related are illustrated in Alg. 1, in which we present a pseudo-code that describes the implementation of a standard version of the PSO algorithm for a global neighborhood definition [5].

**Algorithm 1: PSO**:

1) Initialize a population of particles;
2) Evaluate the desired optimization function at the position given by each particle;
3) Compare the evaluation of each particle with its last best value (the best value obtained so far), called \( p_{\text{best}} \). If current value is better than \( p_{\text{best}} \), adopt it as the new \( p_{\text{best}} \);
4) Compare the current evaluation with the best value visited by the entire population, called \( g_{\text{best}} \). If the current value is better than \( g_{\text{best}} \), adopt it as the new \( g_{\text{best}} \);
5) Update the velocity \( \mathbf{v} \) and the position \( \mathbf{x} \) of the particle according to equations (1) and (2), respectively:
   \[
   \mathbf{v} \leftarrow \mathbf{v} + c_1 (p_{\text{best}} - \mathbf{x}) + c_2 (g_{\text{best}} - \mathbf{x}) \quad (1)
   \]
   \[
   \mathbf{x} \leftarrow \mathbf{x} + \mathbf{v} \quad (2)
   \]
6) Return to step 2) until a stopping criterion is met.

Notice that, in (1), \( c_1 \) and \( c_2 \) are independent random numbers that, in general, are uniformly distributed in the range defined by the interval \([0, 2]\).

After the proposal of the standard PSO algorithm, a number of new proposals were formalized with the aim of improving different aspects, like convergence [6]. In the following, we discuss, in order to give the reader an idea of the development of the research field, a representative set of extensions.

### 2.1 Extensions of the Standard PSO

We now present a list of key extensions to the standard PSO algorithm. The list is far from exhaustive, but includes some of the most widespread methods belonging to the field.

In 1998, Shi and Eberhart [7] proposed the use, within step 5 of Alg. 1, of an inertia weight \( \omega \) as a factor to establish a direct dependence between the present and the updated velocities:

\[
\mathbf{v} = \omega \mathbf{v} + c_1 (p_{\text{best}} - \mathbf{x}) + c_2 (g_{\text{best}} - \mathbf{x}) \quad (3)
\]

being \( c_1 \) and \( c_2 \), again, random numbers as shown in eq. (1). An interesting feature is that larger values of \( \omega \) tend to enhance global search / exploration, whereas smaller values tend to emphasize local search / exploitation [6].

In 1999, Clerc introduced the PSO with convergence agents [8]. The convergence factor allows the particles to oscillate around randomly defined regions related to \( p_{\text{best}} \) and \( g_{\text{best}} \), which may lead to a more efficient exploitation and convergence. The factor and the resulting algorithm are again modifications of step 5 of Alg. 1:

\[
\text{CF} = \frac{2}{2 - \phi - \sqrt{\phi^2 - 4\phi}}, \quad \text{where} \phi = c_1 + c_2 > 4 \quad (4)
\]

\[
\mathbf{v} = \text{CF} \left[ \mathbf{v} + c_1 (p_{\text{best}} - \mathbf{x}) + c_2 (g_{\text{best}} - \mathbf{x}) \right] \quad (5)
\]

with \( c_1 \) and \( c_2 \) defined as in eq. (1) and (3). Experimental results show that, in comparison with the PSO algorithm with inertia weights, the convergence of the algorithm defined in (5) is significantly improved [8].

Among other variations of the PSO and related proposals, we may cite: the idea of Lu and Hou of
introducing auto-adaptation [9]; the operator of differential evolution of Zhang and Xie [10]; the use of Kalman filtering in [11]; the nonlinear dynamical elements exposed in [12] and the use of the concept of natural selection [13]. Finally, it is important to remark that a more systematic view can be found, for instance, in the taxonomy of PSO elaborated by Sedighizadeh and Masehian [14].

3 Meta-Heuristics Inspired in Electromagnetic Theory

Electromagnetic theory is a vast corpus that models, inter alia, the mechanisms underlying charge attraction and repulsion. The algorithm described in this section, proposed in [15], can be understood as an application of this theory to the problem of function optimization.

Basically, each solution to a given problem possesses an associated value - which is defined by evaluating the cost function at the point in question - responsible for defining a sort of “hierarchy” with respect to charge values. This measure determines the forces acting on each relevant point in terms of a version of Coulomb’s law [16], allowing the algorithm to calculate the resultant force, which indicates the direction in which the particle will move.

The algorithm starts by randomly distributing \( m \) points in the search space, with \( n \) dimensions. Then, it evaluates the cost function for each point, being the best solution stored [15]. In the next step, the algorithm defines, for each point, the direction of motion. Finally, the particles move and process is repeated.

The charge of each point \( i \), \( q^i \), determines its capacity of attraction and repulsion. It is calculated using the following expression:

\[
q^i = \exp \left( -n \frac{f(x^i) - f(x^{best})}{\sum_{k=1}^{m} (f(x^k) - f(x^{best}))} \right), \quad \forall i
\]

(6)

Note, in eq. (6), that the point with the best cost has the greatest influence over the others. The charge value receives no signal, and the vector orientation of the force is defined according to eq. (7), which, as already stated, is based on Coulomb’s law:

\[
F^i = \sum_{j=1}^{m} \left( \frac{q^j q^i}{\| x^j - x^i \|^2} , \quad \text{if} \ f(x^i) < f(x^j) \right), \quad \forall i
\]

(7)

The point with the largest (best) cost attracts points with lower values: consequently, the best point attracts all others and the worst point repels them.

The \( i \)-th point is moved according to the direction determined by the resultant force, with a random step-size uniformly distributed between 0 and 1. The step size is bounded by \( u_i \) and \( l_i \), which are, respectively upper and lower bounds:

\[
x^i_k = \begin{cases} 
    x^i_k + \lambda \frac{F^i_k}{\|F^i\|} (u^i_k - x^i_k), \quad \text{if} \ F^i_k > 0 \\
    x^i_k + \lambda \frac{F^i_k}{\|F^i\|} (l^i_k - x^i_k), \quad \text{otherwise}
\end{cases}, \quad (8)
\]

where \( k=1,2,...,n \).

Like PSO, in this metaheuristic, each individual represents a candidate solution, and the population as a whole evolves according to a sort of accumulated collective knowledge. A more detailed description of the algorithm can be found in [15], [17].

4 Differential Evolution

The differential evolution (DE) algorithm was developed by Storn and Price in 1995 [18]. The algorithm starts with the random generation of points in the search space, being each point an individual of a population of solutions to the problem at hand [19]. The population size is a free parameter of the algorithm.

After spreading the individuals over the search space, three individuals are randomly selected. One must have fitness higher than or necessarily equal to that of the current point. These four points, the current and three selected ones, generate a new individual by means of crossover, which is modeled in eq. (9) of Alg. 2 [19].

The next step is mutation, which combines the new and the current individual in accordance with a given probability, which is also a parameter of the algorithm. Finally, the value of the cost function for this final individual is calculated: if it is better than that of the original individual, it acquires the
right of composing the next generation; otherwise, the current individual is kept.

An implementation of the algorithm is shown in Alg. 2.

**Algorithm 2: Differential Evolution**

1) generate N points as initial population
2) while stopping condition is not true do
3) for each point i do
   3.1) randomly select point \( x^d \) such that \( f(x^d) \geq f(x^i) \)
   3.2) randomly select two points, \( x^a \) and \( x^b \)
   \[
   u = (F + 0.5) \ast x^d + (F - 0.5) \ast x^i + F \ast (x^a - x^b) \]  \( (9) \)
   3.3) for each dimension \( j \) do
      if \( \text{rand}() > pr \)
      \[
      P_j = u_j
      \]
      else
      \[
      P_j = x^i_j
      \]
   3.4) if \( f(P) \) is better than \( f(x^i) \)
   add \( P \) to the new population
   else
   add \( x^i \) to the new population

In the above code, \( pr \) is probability of recombination, \( N \) is the population size and \( F \) is a factor that controls the amplification of differential variations.

According to [20], eq. (9) provides an increased balance between exploitation and exploration.

5 Estimation of Distribution Algorithm

The Estimation of Distribution Algorithm (EDA), also known as Probabilistic Model Building Genetic Algorithm (PMBGA) [21], is characterized by being capable of detecting patterns of evolution in the population through the application of probabilistic models. As usual, each individual represents a candidate solution [22], and the initial population is, in general, randomly generated.

After the algorithm evaluates the cost function at each point of the population, a sample is obtained by selecting a subset of promising points. A probabilistic model is then applied to this sample in order that promising areas be estimated to guide the generation of new sample points. The following pseudo-code describes in a systematic way the operation of the algorithm.

**Algorithm 3: Estimation of Distribution Algorithm**

1) randomly generate \( N \) individuals over the search space
2) while stopping condition is not true do
   2.1) evaluate population
   2.2) select \( M \) promising solutions.
   2.3) build a probabilistic model from the \( M \) selected individuals.
   2.4) generate new solutions according to the probabilistic model.

In the above code, \( N \) is the population size and \( M \) is the number of individuals selected to generate the probabilistic model.

In this algorithm, it is possible to modify both the number of selected individuals and the way this selection occurs. Distinct possibilities related to the latter aspect give rise to a number of solutions:

- Factorized Distribution Algorithm (FDA) [23]
- Bayesian Optimization Algorithm (BOA) [24]
- Univariate Marginal Distribution Algorithm (UMDA) [25]
- Population-Based Incremental Learning (PBIL) [26]
- Mutual information maximization for input clustering algorithm (MIMIC) [27]

Information on the general characteristics of the solutions can facilitate the choice of the probability model. Other important factors are the complexity of the employed model, the overall computational cost of processing and the question of storage.

6 DE / EDA Algorithm

In order to aggregate information obtained by the EDA and global differential information ob-
tained by DE, an attempt of combining them into a single method – the DE / EDA [19] - was developed. The main features of this method are exposed with the aid of the pseudo-code shown in Alg. 4.

Algorithm 4: DE / EDA

1) randomly generate N individuals over the search space
2) while the stopping condition is not true do
   2.1) evaluate population
   2.2) select M promising solutions.
   2.3) build a probabilistic model Prob(µ, σ) from the M selected individuals.
   2.4) for each individual i do
       2.4.1) randomly select point xd such that f(xd) ≥ f(xi)
       2.4.2) randomly select two points x^a and x^b
       2.4.3) for each dimension j do
           if rand() > L
               \[ P_j = \frac{x^d_j + x^a_j}{2} + F \cdot (x^d_j - x^a_j + x^a_j - x^b_j) \]  \hspace{1cm} (10)
           else
               \[ P_j = \text{Prob}(\mu_j, \sigma_j) \]
       2.5) if f(P) is better than f(x^i)
           add P to the new population
       else
           add x^i to the new population

In the above code, Prob(µ, σ) represents the probability distribution model and L is used to balance the contributions from the global information and the differential information.

7 Magnetic Particle Swarm Optimization

After having presented a number of approaches that are relevant as historical milestones and that will also play a key role as benchmarks, in this section, we reach the contribution that motivates this work: the Magnetic Particle Swarm Optimization (MPSO) algorithm [1]. We will start our discussion from an analysis of the motivation underlying the proposal.

7.1 Motivation

The physical motivation of the proposed method lies in certain elements of the behavior of magnetic dipoles\(^1\). The aim, as will be discussed in the following, is to seek elements potentially capable of improving the exploration carried out by the algorithm [1].

In a certain sense, the MPSO is an attempt to bring together “good” features of certain metaheuristics, not from the nave standpoint of building a “method of methods”, an idea clearly in conflict with the essence of no free lunch theorems [28], but simply to reach a modus operandi suitable to deal with complex multimodal optimization tasks. Some features taken into account were:

- Populational character.
- Existence of a radius of influence for an individual.
- Balance between exploration and exploitation.
- Iterativeness and memory.

These features are part of strategies belonging to a wide range of approaches: PSO, genetic algorithms, artificial immune systems, ant colony approaches etc.

7.2 The Algorithm

The MPSO starts by distributing the particles uniformly along the search space and calculating the radius of the repulsion region of each of them [1, 2]. The maximum and minimum bounds associated with these radii are calculated based on the number of particles and the size of the effective search space. The minimum radius is typically set as the thousandth part of the maximum radius, this (non-mandatory) choice having been supported by some empirical analyses.

\(^1\)Eventually, the dipoles might be referred to as “charges” or “particles”, without, however, any subjacent allusion to the concept of magnetic monopole.
The repulsion region has the relevant role of preventing the particles from becoming excessively close to one another. Nothing happens when there is an intersection exclusively between regions, but, when a particle invades another’s repulsion region, the worst of them (in terms of the cost function) is removed therefrom. The size of each repulsion region is calculated during execution (obeying the aforementioned bounds) in terms of cost values: when a particle has its cost improved, its repulsion region increases, decreasing otherwise. Initially, the particles move according to equations (11), (12) and (13), three points being generated per iteration:

\[ P_{aux1k} = P^i_k + \lambda^i_k \text{RepRegion}_i \]  
\[ P_{aux2} = P^i + \lambda P^i \]  
\[ P_{aux3} = P^i + \lambda \text{RepRegion}_i \]  

where \( k = 1, \ldots, n \) and \( i=1,\ldots, m \), being \( n \) the number of dimensions and \( m \) the population size. \( \text{RepRegion}_i \) is the repulsion region of particle \( i \), \( \lambda \) is a random number belonging to \([-1,1]\), and \( \lambda^i_k \) is calculated for each dimension \( n \).

When a better solution is found, the particle stores it and the direction along which it has moved. In the subsequent step, the algorithm generates the new point on this line. If the generated point is worse than the current solution, two perpendicular points are studied. The best of these points can be adopted if it is better than the current point, and, in this case, the perpendicular line will also be adopted. If any point along the perpendicular line is better than the current solution, the particle “loses the direction” and the new point is randomly generated in the repulsion region. In this case, there is a decrease in the repulsion region until the algorithm finds a better point than the current one.

If the particle does not find better points, the radius of the repulsion region associated therewith is decreased until the minimum value is reached. As mentioned earlier, if a particle enters another particle’s region, the particle with lowest cost value will be taken away along the line associated with the better particle. The size of this position change must be enough to place the particle outside the repulsion region.

On the other hand, if the best particle has no direction to follow, the worst particle uses its own direction, with a proper step size, to leave the region. If neither has an associated direction, the worst particle will be sent towards the best particle. There ideas are clarified and detailed with the aid of Alg. 5.

**Algorithm 5:** Magnetic Particle Swarm Algorithm

**Initialize()**

while termination criteria are not satisfied do

Move particles()

Verify confronts()

end while

Let us now consider in more detail each function present in the above pseudo-code.

**Initialize()**

This function distributes the particles in the space of feasible solutions in accordance with a uniform distribution, and calculates the upper and the lower bounds of the repulsion regions. The upper bound of the repulsion region (\( ubrr \)) is:

\[ ubrr^k = \frac{(u^k - l^k)}{2 \left( \frac{m}{n} \right)} \]  

being \( u^k \) the upper bound of the search space in dimension \( k \), and \( l^k \) the lower bound of the search space in dimension \( k \), \( n \) the number of dimensions, \( k = 1, \ldots, n \), and \( m \) the number of particles. The lower bound of the repulsion region (\( lbrr \)) is:

\[ lbrr^k = 0.001 \cdot ubrr^k \]  

**Move particles()**

for each particle do

if (particle has a direction)

generate one point in this direction

if (the new point is better)

store this point

increase in 10% the repulsion region limited by \( ubrr \)

else

generate two perpendicular points

if (the best of them is better than current point)
store the best of them and its direction
else
lose direction
end if
end if
else
generate three points using eqs. (11)(12)(13)
if( the best generated point is better than the current point)
store the best generated point and its direction
else
if( never had a direction)
store best generated point
else
decrease in 1% the repulsion region limited by lbrr
end if
end if
end if
end for

Verify confronts()
for each particle pair do
if(distance between pair is less than the largest repulsion region)
if( the better particle has a direction)
move the worse particle in this direction
else
if( the worse particle has a direction)
move it in this direction
else
move the worse particle in the direction of the best particle.
end if
end if
end if
end for

7.3 Magnetic Particle Swarm Optimization with Estimation of Distribution

The promising behavior of the MPSO in a number of tests described in [1] led to the proposal of a variant: the MPSO with estimation of distribution [2]. The motivation was essentially to include a step of estimation of distribution to increase the ability of the algorithm in finding promising search patterns from the information contained in the population [2].

The estimation step is periodically applied in order to generate a number of individuals that replace the worst representatives of the population. An improvement in terms of both exploration and convergence is thus expected.

Algorithm 6: Magnetic Particle Swarm Algorithm (with estimation of distribution)

initialize()
while termination criteria are not satisfied do
  Move particles()
  Verify confronts()
  if(cycles are multiple of T)
    Estimation of Distribution()
  end if
end while

The function Estimation of Distribution, which forms the core of the variant, is detailed in the following:

Estimation of Distribution()

using all particles, calculate the mean and standard deviation for each dimension.
generate K new particles with a normal distribution.
include these K new particles in the population, replacing the K worst particles.
Select the M best particles.

8 Experiments and Results

To analyze the performance of the MPSO algorithm and of the MPSO with estimation of distribution, bases for comparison were established, for a set of representative benchmark functions, with the
following methods: the PSO algorithm (vide section 2), an electromagnetic algorithm (vide section 3) and DE with estimation of distribution (vide sections 5 and 6). The employed cost functions will now be discussed in detail.

8.1 Cost Functions

All algorithms were tested in the context of eight optimization problems defined by cost functions with distinct characteristics, all of them well-known in the literature [29].

1) Sphere Function

\[ f(x) = \sum_{i=1}^{n} x_i^2, \]  

where \( x \in [-100,100]^n \). The global minimum is located at \( x^*=[0,0,\ldots,0] \).

2) Rosenbrock Function

\[ f(x) = \sum_{i=1}^{n-1} 100(x_{i+1}^2 - x_i^2)^2 + (x_i - 1)^2, \]  

where \( x \in [-30,30]^n \) and the global optimum is \( x^*=[1,1,\ldots,1] \).

3) Griewank Function

\[ f(x) = 1 + 0.00025 \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos \frac{x_i}{\sqrt{i}}, \]  

where \( x \in [-600,600]^n \). The global optimum is \( x^*=[0,0,\ldots,0] \).

4) Rastrigin Function

\[ f(x) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)], \]  

where \( x \in [-5.12, 5.12]^n \) and the global optimum is \( x^*=[0,0,\ldots,0] \).

5) Salomon Function

\[ f(x) = 1 - \cos \left(2\pi \sqrt{\sum_{i=1}^{n} x_i^2}\right) + 0.1 \sqrt{\sum_{i=1}^{n} x_i^2}, \]  

where \( x \in [-100,100]^n \) and the optimum is \( x^*=[0,0,\ldots,0] \).

6) Schwefel Function

\[ f(x) = -\sum_{i=1}^{n} x_i \sin \sqrt{|x_i|}, \]  

where \( x \in [-500,500]^n \) and the global optimum is \( x^*=[420.97,\ldots,420.97] \).

7) Levy Function

\[ f(x) = \sin^2 (\pi y_1) + \sum_{i=1}^{n-1} \left[ (y_i - 1)^2 (1 + 10\sin^2 (\pi y_i + 1)) \right] + (y_n - 1)^2 (10\sin^2 (2\pi y_n)), \]  

where \( y_i = 1 + \frac{(x_i - 1)}{4} \), \( x \in [-10,10]^n \) and the global optimum is \( x^*=[1,1,\ldots,1] \).

8) F1 Function

\[ f(x) = \sum_{i=1}^{n} |x_i - i|, \]  

where \( x \in [-100, n+1]^n \) and the global optimum is \( x^*=[1,2,\ldots,n] \).

The obtained results will now be exposed and discussed. It is important to highlight that they include tests shown in this work for the first time, like analysis the number of particles shown in Tab. II.

8.2 Results

In the experiments, all the selected algorithms have their performances compared with those of both versions of the MPSO algorithm. To allow proper bases for comparison, all methods made use of 30 particles (individuals), one per dimension, and were allowed 1.000.000 (one million) cost function evaluations. Similar settings were established for analogous methods, and all parameters were chosen in accordance with several preliminary tests. All algorithms were run 30 times for each case, and the mean and standard deviation of the attained costs were calculated.

The PSO used \( c_1 = 0.5 \) and \( c_2 = 0.2 \) for all functions. The DE / EDA algorithm used a normal distribution and balance of contribution \( L = 50\% \), size of population = 30 and scaling factor \( F=0.6 \) for all functions. The MPSO with ED executed the ED
step every 100 cycles, being the mean and standard deviation calculated with the best five particles, and a new particle was generated replacing the worst current particle. The results are summarized in Tab. I, being the best performance for each problem boldfaced.

Note that one of the versions of the MPSO reaches the best performance for seven out of eight problems, being the method with estimation of distribution, in the case of the F1 function, second only to the DE/EDA. It is interesting to notice that, on the other hand, the standard MPSO algorithm had a better performance for the Salomon cost function. This seems to suggest that the inclusion of the estimation of distribution step may have a particularly auspicious effect on the convergence for monomodal scenarios.

The set composed of Figs. 1 – 8, shown for the first time in this work, contains the time evolution of the cost associated with the best particle and the average cost of the population for different functions. In all cases, the behavior is derived from the execution of the standard MPSO.

**Figure 1.** Time evolution - cost associated with the best particle.

**Figure 2.** Time evolution – average cost.

**Figure 3.** Time evolution - cost associated with the best particle.

**Figure 4.** Time evolution – average cost.

**Figure 5.** Time evolution - cost associated with the best particle.

**Figure 6.** Time evolution – average cost.
**Table 1.** Mean and standard deviation values of the best cost value – Search Space with 30 dimensions.

<table>
<thead>
<tr>
<th>Functions</th>
<th>Algorithms</th>
<th>Global</th>
<th>PSO</th>
<th>Eletromagnetic</th>
<th>DE/EDA</th>
<th>Magnetic Particle Swarm</th>
<th>Magnetic Particle Swarm with ED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere</td>
<td>0.00</td>
<td>0.02±0.03</td>
<td>0.26±0.05</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td></td>
</tr>
<tr>
<td>Rosenbrock</td>
<td>0.00</td>
<td>28.98±0.05</td>
<td>83.02±88.95</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td></td>
</tr>
<tr>
<td>Griewank</td>
<td>0.00</td>
<td>0.05±0.08</td>
<td>0.55±0.09</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td></td>
</tr>
<tr>
<td>Rastrigin</td>
<td>0.00</td>
<td>0.01±0.02</td>
<td>45.77±17.59</td>
<td>73.60±8.82</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td></td>
</tr>
<tr>
<td>Salomon</td>
<td>0.00</td>
<td>0.10±0.00</td>
<td>1.38±0.08</td>
<td>0.12±0.04</td>
<td>0.00±0.00</td>
<td>0.09±0.02</td>
<td></td>
</tr>
<tr>
<td>Schwefel</td>
<td>-12569.49</td>
<td>-9636.93±709.21</td>
<td>-8606.34±705.50</td>
<td>-10762.50±885.11</td>
<td>-12569.42±0.09</td>
<td>-12569.42±0.09</td>
<td></td>
</tr>
<tr>
<td>Levy</td>
<td>0.00</td>
<td>1.44±3.12</td>
<td>0.32±0.31</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td>0.00±0.00</td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>0.00</td>
<td>311.70±146.23</td>
<td>32.26±16.79</td>
<td>0.00±0.00</td>
<td>8.10±1.82</td>
<td>5.91±1.42</td>
<td></td>
</tr>
</tbody>
</table>

The information contained in these figures can be straightforwardly interpreted: the convergence of the best individual is always relatively fast (having in view the allowed number of function evaluations), and the average cost changes (with more or less oscillation) in a much slower pace, which means that population diversity is maintained. This confirms that the motivations underlying the building of the method were, to some extent, justified.

In order to bring some elements of a sensitivity analysis, we analyze the MPSO for the F1 cost function under different numbers of particles (always keeping the maximum number of cost evaluations): the results are shown in Tab. II. The choice of the F1 function is due to the fact that it led to the worst results for the MPSO in the tests described in Tab. I.

![Figure 7. Time evolution - cost associated with the best particle.](image7.jpg)

![Figure 8. Time evolution – average cost.](image8.jpg)
Table 2. Mean and standard deviation values of the best cost value – Search Space with 30 dimensions.

<table>
<thead>
<tr>
<th>Functions</th>
<th>Algorithm – MPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of Particles</td>
</tr>
<tr>
<td>F1</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 3. Mean and standard deviation values of the best cost value – Search Space with 30 dimensions.

<table>
<thead>
<tr>
<th>Functions</th>
<th>Algorithm - MPSO - 5 particles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Global</td>
</tr>
<tr>
<td>F1</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 4. Mean and standard deviation values of the best Cost Value search space with 1000 dimensions.

<table>
<thead>
<tr>
<th>Functions</th>
<th>Algorithm Magnetic Particle Swarm</th>
<th>DE/EDA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Global</td>
<td>Mean ± Standard Deviation</td>
</tr>
<tr>
<td>Sphere</td>
<td>0.00</td>
<td>0.00 ± 0.00</td>
</tr>
<tr>
<td>Rosenbrock</td>
<td>0.00</td>
<td>0.00 ± 0.00</td>
</tr>
<tr>
<td>Griewank</td>
<td>0.00</td>
<td>0.00 ± 0.00</td>
</tr>
<tr>
<td>Rastrigin</td>
<td>0.00</td>
<td>0.00 ± 0.00</td>
</tr>
<tr>
<td>Salomon</td>
<td>0.00</td>
<td>0.00 ± 0.00</td>
</tr>
<tr>
<td>Schwefel</td>
<td>-418982.9</td>
<td>-418935.53 ± 34.95</td>
</tr>
<tr>
<td>Levy</td>
<td>0.00</td>
<td>0.03 ± 0.03</td>
</tr>
<tr>
<td>F1</td>
<td>0.00</td>
<td>144046.11 ± 1657.31</td>
</tr>
</tbody>
</table>
Note that the standard MPSO reaches the best performance for 5 particles, which is rather interesting in view of the search space dimension. Probably this relatively reduced number of particles is related to the monomodal character of the F1 function, which is illustrated, for a 3D case, in Fig 9. In such a case, an increase in the number of particles might cause undesired disturbance in the pattern of convergence towards the single minimum.

In Tab. III, the results of another sensitivity test are shown. Now, the number of cost function evaluations is modified, being the other parameters kept equal. Once again, the F1 function is employed.

Note that an increase in the number of evaluations revealed that there was room for improvement in the performance of the method, although the excellent performance attained via differential evolution was not reached. This confirms that the multimodal search character is predominant for the structure that characterizes the MPSO algorithms.

The last test was to deal with an elevated number of dimensions: one thousand. The performance of the standard MPSO is remarkable: it performed better that the DE/EDA method in all cases, except, once more, for F1. This result confirms the potential of the methodology and the “F1 exception” perhaps can be directly understood as a corollary to the no free lunch theorems [28].

9 Conclusion

In this work, we analyzed the novel Magnetic Particle Swarm Optimization (MPSO) paradigm by means of the two existing representatives thereof: the standard MPSO [1] and the variant called MPSO with estimation of distribution [2]. From a conceptual standpoint, the paradigm combines ideas derived from electromagnetic theory with general notions belonging to well-established global optimizers.

The new algorithms were tested for a representative set of benchmark functions and proved themselves to be very promising search tools, particularly in multimodal domains. It should be remarked that the results presented in this work comprise, for the first time, elements of convergence and sensitivity analyses.

Perspectives for future work include performing a more extensive sensitivity analysis and extending the scope of comparison both in terms of benchmark functions (which may include some examples of practical applications) and of metaheuristics.

10 Acknowledgement

The authors thank CAPES for the financial support.

References


Conclusion

thereof: the standard MPSO [1] and the variant free lunch theorems [28].

The performance structure that characterizes the MPSO algorithms. multimodal search character is predominant for the evolution was not reached. This confirms that the excellent performance attained via differential evolutions revealed that there was room for improve-

can be directly understood as a corollary to the no number of dimensions: one thousand. The performance of the method, although in the performance of the no-

Once again, the F1 function is employed. Now, the number of cost function eval-

The last test was to deal with an elevated num-

Note that the standard MPSO reaches the best of practical applications) and of metaheuristics. mark functions (which may include some examples in the scope of comparison both in terms of bench-

References


