

A New Multiagent Algorithm for Dynamic Continuous Optimization

Julien Lepagnot, Université de Paris 12, France

Amir Nakib, Université de Paris 12, France

Hamouche Oulhadj, Université de Paris 12, France

Patrick Siarry, Université de Paris 12, France

ABSTRACT

Many real-world problems are dynamic and require an optimization algorithm that is able to continuously track a changing optimum over time. In this article, a new multiagent algorithm is proposed to solve dynamic problems. This algorithm is based on multiple trajectory searches and saving the optima found to use them when a change is detected in the environment. The proposed algorithm is analyzed using the Moving Peaks Benchmark, and its performances are compared to competing dynamic optimization algorithms on several instances of this benchmark. The obtained results show the efficiency of the proposed algorithm, even in multimodal environments.

Keywords: Dynamic, Metaheuristic, Moving Peaks Benchmark, Multiagent, Multimodal Environment, Non-Stationary, Optimization, Time-Varying

1. INTRODUCTION

Recently, optimization in dynamic environments has attracted a growing interest, due to its practical relevance. Many real-world problems are dynamic optimization problems (DOPs), i.e. their objective function changes over time: typical examples are in vehicle routing (Larsen, 2000), inventory management (Minner, 2003) and scheduling (Branke & Mattfeld, 2005). For dynamic environments, the goal is not only to

locate the optimum, but to follow it as closely as possible. A dynamic optimization problem can be expressed by:

$$\begin{aligned} \max \quad & f(\vec{x}, t) \\ \text{s.t.} \quad & h_j(\vec{x}, t) = 0 \text{ for } j = 1, 2, \dots, p \\ & g_k(\vec{x}, t) \leq 0 \text{ for } k = 1, 2, \dots, l \\ & \vec{x} = [x_1, x_2, \dots, x_n] \end{aligned} \quad (1)$$

where $f(\vec{x}, t)$ is the objective function of the problem, $h_j(\vec{x}, t)$ denotes the j^{th} equality

constraint and $g_k(\vec{x}, t)$ denotes the k^{th} inequality constraint. Both of them may change over time, denoted by t . In this article, we focus on a dynamic optimization problem with time constant constraints.

The main approaches to deal with DOPs can be classified into the following five groups (Jin & Branke, 2005):

1. **Generated diversity after a change:** When a change in the environment is detected, explicit actions are taken to increase diversity and to facilitate the shift to the new optimum.
2. **Maintained diversity throughout the run:** Convergence is avoided all the time and it is hoped that a spread-out population can adapt to change more efficiently.
3. **Memory-based approaches:** These methods are supplied with a memory to be able to recall useful information from the past. In practice, they store good solutions in order to reuse them when a change is detected.
4. **Multipopulation approaches:** Dividing up the population into several subpopulations, distributed on different optima, allows tracking of multiple optima simultaneously and increases the probability to find new ones.
5. **Future prediction:** Recently, another kind of methods, trying to predict future changes, has attracted much attention (Rossi, Abderrahim, & Diaz, 2008), (Rossi, Barrientos, & Cerro, 2007), (Simoes & Costa, 2008). This approach is based on the fact that in a real problem, changes can follow some pattern that could be learned.

We focus on population-based metaheuristics, which are global search, generally bio-inspired, algorithms. We can roughly classify them in four categories: evolutionary algorithms (EAs), particle swarm optimization (PSO), ant colony optimization (ACO) and hybrid methods. We will now describe dynamic metaheuristics that have been proposed in the literature in each of these categories.

A lot of existing dynamic optimization methods are EAs. EAs can be indeed well suited for optimization in changing environments, since they are inspired by the principles of natural evolution, and natural evolution deals very well with environmental changes. In (Rossi, Abderrahim, & Diaz, 2008), the algorithm incorporates a motion prediction technique based on Kalman filter, in order to improve the speed of optimum tracking. This kind of prediction could be suited for many problems, but it increases the complexity of the algorithm and the number of parameters. In (Tinos & Yang, 2007), the authors propose to maintain diversity in a genetic algorithm (GA) by replacing the worst individual and some others with randomly generated individuals, and maintaining their offspring in a subpopulation. However, this method is only designed for discrete problems (DDOPs). Another GA is described in (Yang, 2006), that is based on immune system and also devoted to DDOPs. In (Huang & Rocha, 2005), an agent-based coevolutionary GA is proposed. This algorithm makes coevolve the way the genotype of an agent is read along with its genotype. Thus, the chromosomes are transcribed into their “edited” counterparts, using the “editors” which coevolve with them, then crossed-over and mutated. In (Mendes & Mohais, 2005), a multipopulation differential evolution (DE) algorithm is proposed, in which some techniques are added in order to increase diversity. DE is a population-based approach, its strategy consists in generating a new position for an individual according to the differences calculated between other randomly selected individuals. This algorithm is based on two main parameters, that must be correctly fitted. However in (Mendes & Mohais, 2005), these parameters are randomly generated in order to make DE easier to use.

Another widely used class of algorithms for dynamic optimization is PSO. PSO is a population-based approach, similar in some respects to EAs, except that potential solutions (particles) move, rather than evolve, throughout the search space (Blackwell & Branke, 2004). The move rules, or particle dynamics, are inspired by

models of swarming and flocking (Kennedy & Eberhart, 1995). In this class of methods, we can cite (Blackwell & Branke, 2004), where the authors propose two multi-swarm algorithms based on an atomic model. The first algorithm uses multiple swarms, composed of a sub-swarm of mutually repelling particles, orbiting around another sub-swarm of neutral, or conventional PSO particles. The second algorithm is based on a quantum model of the atom, where the charged particles (electrons) will not follow a classical trajectory, but will be rather randomized within a ball centered on the swarm attractor. Both of these algorithms place their swarms on each of localized optima, thus letting each swarm track a different optimum. However, these methods have many parameters that must be suitably fitted. Moreover, the obtained results with these methods are not good enough. This approach of using charged particles is also used in (Blackwell & Branke, 2006) and (Li, Branke, & Blackwell, 2006), with other techniques to increase diversity and to track optima. Another PSO algorithm is proposed in (Du & Li, 2008), that uses two populations of particles. The first one is for diversification, and the second is for intensification.

Another class of algorithms, ACO, has been used for dynamic optimization. ACO is also a population-based approach using swarm intelligence, inspired from the way ants find the shortest path between the nest and a source of food (Dorigo & Gambardella, 2002). In this class of methods, we can cite (Dréo & Siarry, 2006), that proposes to hybridize an ACO algorithm with the Nelder-Mead simplex method (Nelder & Mead, 1965) for local search. A modified Nelder-Mead simplex method is also developed in this article for dynamic optimization. However, this method does not perform well on functions having a lot of local optima. In (Tfaily & Siarry, 2008), charged ants are used in order to maintain diversity. Moreover, the attribution of an electrostatic charge to each ant prevents them from converging to same local optima. The performances of this method are not good enough, according to the benchmark defined in (Dréo & Siarry, 2006).

In order to perform better on dynamic problems, some authors have tried hybrid methods, like in (Lung & Dumitrescu, 2007) and (Lung & Dumitrescu, 2008), that propose hybrid PSO/EAs algorithms. However, the results of those methods are not so different from PSO or EAs methods, described previously.

An algorithm proposed in (Moser & Hendtlass, 2007) is based on extremal optimization (EO). EO does not use a population of solutions, but improves a single solution using mutation. This algorithm uses a “stepwise” sampling scheme that samples every dimension of the search space in equal distances. Then, the algorithm takes the best candidate as the next solution, afterwards it proceeds to a hill-climbing phase, in order to fine-tune the solution. Then, the solution is stored in memory, and the method is applied again on another randomly generated solution. This algorithm is simple and efficient, and allows to obtain good results on a specific test called the “Moving Peaks Benchmark” (MPB) (Branke, 1999). However, this method is especially developed and fitted for this benchmark, and it seems not applicable in real world problems.

In this article, a new method is proposed to solve dynamic optimization problems. Our algorithm, called “MADO”, for “MultiAgent Dynamic Optimization”, has been developed in order to be efficient for solving a wide range of DOPs. It is a multiagent method, that makes use of a population of agents to explore the search space. The proposed algorithm is based on the following considerations: when optimizing in a multimodal environment, we need to keep track of each local optimum, to overcome the case where the global optimum “jumps” from one of them to another. The found optima are archived in a memory. Then, this memory can be used when a change is detected.

It is common that real world problems have time costly evaluation of fitness functions. Hence, the computational cost of the proposed algorithm should be made as short as possible and expressed in terms of number of evaluations. The proposed algorithm makes use of the following inspirations:

- Keeping information about the previous positions allows to prevent wasting evaluations in unpromising zones of the search space;
- Using a sampling of candidate solutions that optimally cover the local landscape may reduce the number of evaluations per trajectory step, without decreasing performance. For this goal, we maximize the distances between all the candidate solutions, whereas keeping them constricted into the current local landscape. More precisely, it is done by evaluating the candidate solutions on the surface of an hypersphere centered on the current position;
- The sampling of candidate solutions is adaptable to the local landscape. This is done by using an adaptable radius for the above-mentioned hypersphere;
- An exclusion radius for the current solution of each search trajectory is used to avoid the case where some search trajectories explore the same area of the search space.

The rest of this article is organized as follows. Section 2 presents test problems used in the literature and introduces the benchmark set that will be used here. Section 3 describes the proposed algorithm. Experimental results are discussed in Section 4. Conclusion and works in progress are in Section 5.

2. BENCHMARK SET FOR DYNAMIC OPTIMIZATION

Table 1 presents a summary of competing methods available in the literature and gives the test problems used by each one. The test problems gathered in the first column of Table 1 are those used by the competing methods. These competing methods are described in the references listed in the third column. From Table 1, one can remark that the most commonly used testbed is the Moving Peaks Benchmark (MPB) (Branke, 1999). This benchmark is becoming

the standard for testing dynamic optimization algorithms, and is claimed to be representative of real world problems (Branke, 1999). To compare our algorithm to the competing ones, this testbed was adopted.

MPB consists of a number of peaks that vary their shape, position and height randomly upon time. At any time, one of the local optima becomes the new global optimum. MPB generates DOPs consisting of a set of peaks that periodically move in a random direction, by a fixed amount s (the change “severity”). The movements are autocorrelated by a coefficient $0 \leq \lambda \leq 1$, where 0 means uncorrelated and 1 means highly autocorrelated. The peaks change position every α iterations (function evaluations), and α is called time span. The fitness function for the landscape of MPB is formulated as follows:

$$F(\vec{x}, t) = \max_{i=1, \dots, m} \left(H_i(t) - \left(W_i(t) \sqrt{\sum_{j=1}^d (x_j - X_{ij}(t))^2} \right) \right) \quad (2)$$

where m is the number of peaks, d is the number of dimensions, $H_i(t)$ is the height of the i^{th} peak at the time t , $W_i(t)$ is the width of the i^{th} peak at the time t and $\vec{X}_i(t)$ is the position of the i^{th} peak at the time t .

Figure 1 illustrates an MPB landscape before and after a change (after one time span).

In order to evaluate the performance, the “offline error” is used. Offline error (oe) is defined as the average of the errors of the best points evaluated during each time span. It is defined by:

$$oe = \frac{1}{Nc} \sum_{j=1}^{Nc} \left(\frac{1}{Ne(j)} \sum_{i=1}^{Ne(j)} (f_j^* - f_{ji}^*) \right) \quad (3)$$

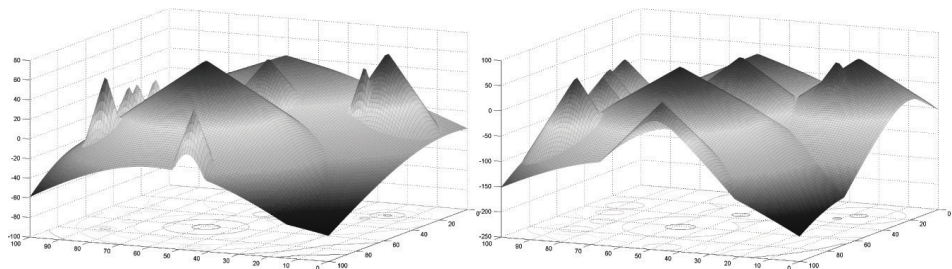
where Nc is the total number of fitness landscape changes within a single experiment, $Ne(j)$ is the number of iterations performed for the j^{th} state of the landscape, f_j^* is the value

Table 1. Summary of recent DOP algorithms

Test problems	Base algorithm	Authors
MPB ¹ and the dynamic Rastrigin function	PSO ²	Du & Li, 2008
various continuous dynamic problems (Dréo & Siarry, 2006)	ACO ³	Tfaily & Siarry, 2008
a computer vision problem	EAs ⁴	Rossi, Abderrahim, & Diaz, 2008
MPB	Hybrid PSO/EAs	Lung & Dumitrescu, 2008 (<i>ESCA</i>)
MPB	Hybrid PSO/EAs	Lung & Dumitrescu, 2007 (<i>CESO</i>)
MPB	EO ⁵	Moser & Hendtlass, 2007
a set of discrete dynamic problems (Yang, 2003), (Yang & Yao, 2005)	GA ⁶	Tinos & Yang, 2007
a set of discrete dynamic problems (Yang, 2003), (Yang & Yao, 2005)	GA	Yang, 2006
various continuous dynamic problems (Dréo & Siarry, 2006)	ACO	Dréo & Siarry, 2006
MPB	PSO	Li, Branke, & Blackwell, 2006
MPB	PSO	Blackwell & Branke, 2006 (<i>2nd version</i>)
MPB	DE ⁷	Mendes & Mohais, 2005
a testbed proposed by the authors	EAs	Huang & Rocha, 2005
MPB	PSO	Blackwell & Branke, 2004 (<i>1st version</i>)

¹ The Moving Peaks Benchmark (Branke, 1999). ² Particle Swarm Optimization. ³ Ant Colony Optimization. ⁴ Evolutionary Algorithms. ⁵ Extremal Optimization. ⁶ Genetic Algorithm. ⁷ Differential Evolution.

Figure 1. An MPB landscape before and after a change



of the optimal solution for the j^{th} landscape and $f_{j_i}^*$ is the current best fitness value found for the j^{th} landscape.

We can remark that this measure has some weaknesses: it is sensitive to the overall height of the landscape, and to the number of peaks. It is important for an algorithm to find the global optimum quickly, to minimize the offline error. Hence, the most successful strategy is a multi-solution approach that keeps track of every local peak (Moser & Hendtlass, 2007).

3. THE PROPOSED MADO ALGORITHM

3.1 Overall Scheme

MADO is a multiagent algorithm that consists in the following three modules:

1. **Memory module:** In case of a multimodal environment, a dynamic optimization method needs to keep track of each local optimum found, since one of them can become the new global optimum after a change. We propose to use a memory to archive the found optima.
2. **Agent manager:** It contains all the agents, and manages their execution, creation and deletion. Agents are nearsighted (they have only a local vision of the search space). More precisely, agents are only performing local search, they jump from their current position to a better one, in their neighborhood, until they cannot improve their current solution, reaching thus a local optimum.
3. **Coordinator:** It counterbalances the nearsightedness of the agents. The coordinator has indeed a global vision of the search performed by the agents, and it is able to prevent them from searching in unpromising zones of the search space. The coordinator supervises the whole search, and

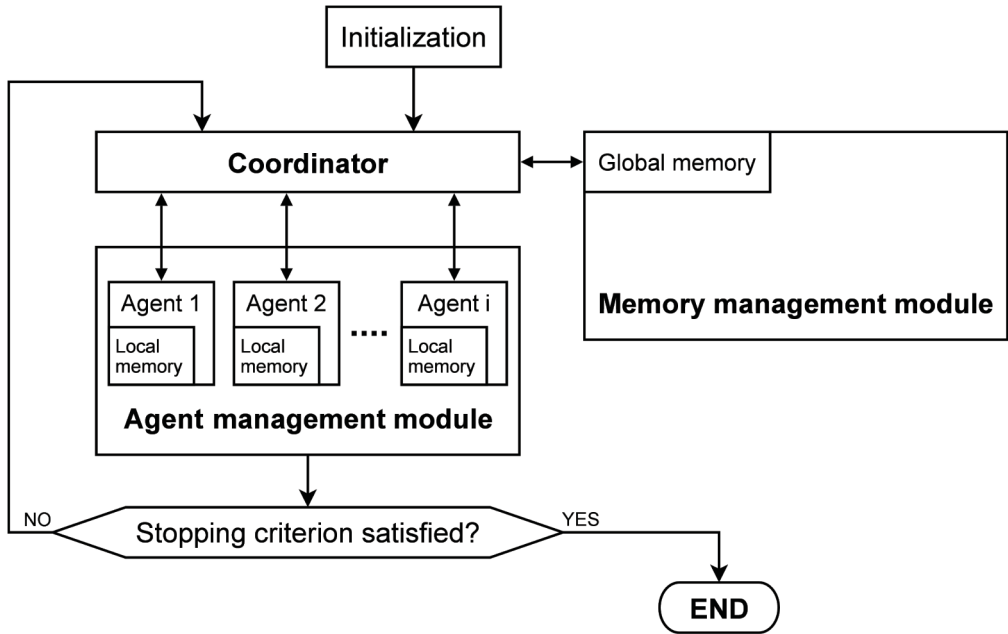
manages the interactions between memory and agents modules.

In the following subsections, a detailed description of the overall scheme of the method is presented. Subsequently, the three modules that compose the MADO algorithm are described. Afterwards, the fitting of the parameters of the algorithm is presented, and their values, further used in the experimental section, are given.

The overall scheme of the MADO algorithm is illustrated in Figure 2. As it is shown, all the interactions between the memory manager and the agent manager are through the coordinator. Moreover, all global decisions are taken by the coordinator. The memory manager maintains the archive of local optima, that are provided by the coordinator. The agent manager informs the coordinator about the found optima, and receives its instructions for creating, deleting, and repositioning agents.

Before explaining in detail these modules, we need to describe how the distances are calculated in MADO. As we are considering an Euclidian space of d dimensions as the search space, the Euclidian distance will be used. However, the search space may not have the same bounds on each dimension. Then, we will use a “normalized” basis. We denote by Δ_i the size of each interval that defines the search space, with $i \in [1, d]$. Then, the unit vectors \vec{e}_i in the direction of each axis of the Cartesian coordinates system are “scaled” in order to produce modified unit vectors u_i , that make all the Δ_i equal to 1 in the basis defined by these modified unit vectors. This is done by using $\{\vec{u}_1 = \frac{\vec{e}_1}{\Delta_1}, \vec{u}_2 = \frac{\vec{e}_2}{\Delta_2}, \dots, \vec{u}_d = \frac{\vec{e}_d}{\Delta_d}\}$ as the basis of the Euclidian space where the search space is defined. Consequently, when an hypersphere inside the search space is considered, it will correspond to an ellipsoid in the canonical basis.

Figure 2. Overall scheme of MADO algorithm



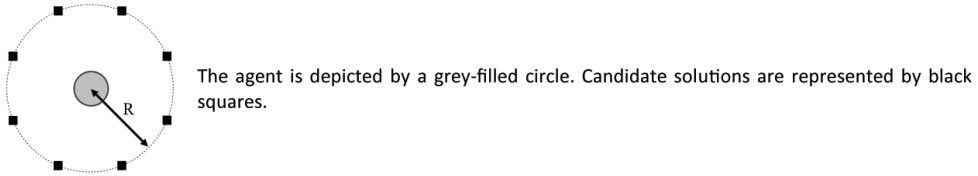
3.2 Agent Management Module

3.2.1 The Exploration Strategy of the Agents

Agents explore the search space step-by-step, moving from their current position to a better one in their neighborhood, until they reach a local optimum. Hence, to precisely describe the behavior of the agents, we first explain the used kind of neighborhood, and how the agents make use of it. As agents are nearsighted, they can only test candidate solutions for their next move in a delimited zone of the search space, centered on their current position. This zone must be bounded. Without any information about the local landscape of an agent, a good choice is to make it isotropic. Then, the most adapted topology is a ball. To keep small the number of fitness function evaluations, we need a sampling that optimally covers the local landscape. Afterwards, we maximize the distances between all candidate solutions inside this

ball. It leads to sample them on the boundary of the ball (on a hypersphere centered on the current solution of an agent). This delimited zone of the search space must be adaptive to the local landscape, to increase the efficiency of the agents. It is done by using an adaptive radius for the ball that defines this zone. From all these considerations, we define the neighborhood of the agents as a set of N candidate solutions placed on the hypersphere of radius R centered on the current position of an agent (N is a parameter of the algorithm). Figure 3 illustrates this neighborhood in dimension 2. Another set of N points are generated and stored at the beginning of the MADO algorithm. These points are calculated in order to meet the following constraints:

- The smallest distance between them is maximized;
- The distances between them and the origin of the space (the point $(0, 0, \dots, 0)$) are equal to 1 ;

Figure 3. The sampling of candidate solutions of an agent, when $d = 2$ and $N = 8$ 

- The number of dimensions of the space in which they are defined match the number of dimensions of the search space.

This computation is done using the well known electrostatic repulsion heuristic (Conway & A., 1998), that considers points as charged particles repelling each other. Starting from an arbitrary distribution, it uses point repulsion, where all points are considered to repel each other according to a $1/r^2$ force law, with r the distance between two points, and dynamics are simulated. The algorithm runs until the satisfaction of a stopping criterion, and the resulting set of points is returned. We use a stopping criterion that is satisfied when no point can be moved by a distance greater than ε , typically equal to 10^{-4} . Then, this set of points is used by agents to get the positions of the candidate

solutions of their current neighborhood. The procedure used to get the exact positions of candidate solutions is presented in Algorithm 1; where S' is the computed set of an agent's candidate solutions, d is the number of dimensions of the search space, S is the precalculated set of uniformly spaced N points on the unit hypersphere and \vec{P}_c is the current position of the agent. We can note that, for $d = 1$, the only possible set of points for S is $\{(-1), (1)\}$ and agents can only move forward or backward by steps of length equal to R . In this particular case, the generated unit vector \vec{V} can only be (-1) or (1) . The use of the Householder reflection (Householder, 1958) is required here in order to sample candidate solutions on the whole surface of the unit hypersphere. It allows generating from S new sets of uniformly spaced

Algorithm 1. Calculates the set of candidate solutions of an agent

```

 $S' \leftarrow \emptyset$ 
 $\vec{V} \leftarrow$  a random unit vector uniformly distributed on the hypersphere of radius
1, of  $d$  dimensions and centered on the origin
foreach point  $\vec{P}_i \in S$  do
     $\vec{P}_i^r \leftarrow$  the Householder reflection of  $\vec{P}_i$  which uses  $\vec{V}$  as the Householder vector
    (to randomly reflect  $\vec{P}_i$ )
     $\vec{P}_i^{rs} \leftarrow \vec{P}_i^r \cdot R$  (to scale  $\vec{P}_i^r$  by  $R$ )
     $\vec{P}_i^{rst} \leftarrow \vec{P}_i^{rs} + \vec{P}_c$  (to translate  $\vec{P}_i^{rs}$  on the agent's local landscape hypersphere)
    if  $\vec{P}_i^{rst}$  is inside the search space then (if it is not beyond the boundary of
    the search space)
         $S' \leftarrow S' \cup \{\vec{P}_i^{rst}\}$ 
    end
end
return  $S'$ 

```

points on this hypersphere, rather than directly using the predefined set S . Moreover, the use of the precalculated set S allows to perform the electrostatic repulsion heuristic only one time (at the beginning of the algorithm), rather than each time a new set of candidate solutions is generated. This way, it saves a significant amount of computation time.

3.2.2 The Step Size Adaptation Strategy

The adaptation of the radius R makes use of trajectory information gathered along the steps of an agent. We propose to use the “cumulative path length control” described in (Hansen & Ostermeier, 2001), with much simpler calculations. The Figure 4 illustrates the two possible kinds of “bad” trajectories that an agent may follow. When one of these cases is detected, an increase or a decrease of the step size R is performed. The first case (Figure 4(a)) may mean that the agent is turning around an optimum, without being able to reach it directly. This is due to a too large step size, leading to back and forth displacements, which cancel each other out. Thus, in this first case, a decrease in R is needed. On the contrary, the displacements on Figure 4(b) are oriented in the same direction, which may

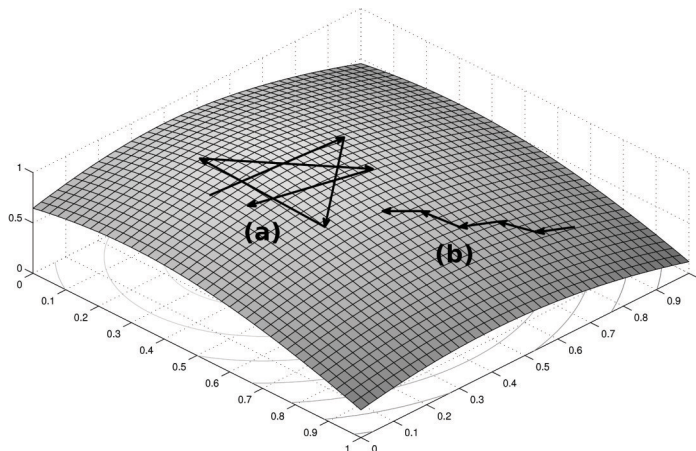
mean that the agent is hill climbing a large peak (a peak is considered to be large when, climbed by an agent, the statement $U_n > s_u$ becomes true (see equation (5) and Algorithm 2)). Then, to avoid too slow moves to the top of this peak, an increase in R is performed.

Information about the successive moves of an agent is collected by using a “cumulative dot product” of old successive displacement vectors. It is done using the following formula:

$$U_n = \begin{cases} \sum_{i=1}^{n-2} \left(c_u \right)^{n-i-2} \frac{\vec{D}_{i,i+1} \cdot \vec{D}_{i+1,i+2}}{\|\vec{D}_{i,i+1}\| \|\vec{D}_{i+1,i+2}\|} & \text{if } n > 2 \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

where U_n is the cumulative dot product of an agent’s search trajectory, n is the number of successive positions of this trajectory, $\vec{D}_{i,i+1}$ is the displacement vector from the i^{th} position to the $(i + 1)^{th}$ position in this trajectory, and c_u is a constant in $[0, 1]$, where 0 gives no weight to displacements before the last one, whereas 1 applies the same weight to the last displacement and to its previous ones. In practice, U_n

Figure 4. The two kinds of trajectories that lead to a step size adaptation



(a) Agent turning around an optimum. (b) Agent hill climbing a large peak.

is computed using a recurrence relation on U_{n-1} and on the two last displacements (see equation (5)). There is no need to record all the displacement vectors of the agents, but only their two last ones.

$$U_n = \begin{cases} c_u U_{n-1} + \frac{\vec{D}_{n-2,n-1} \cdot \vec{D}_{n-1,n}}{\|\vec{D}_{n-2,n-1}\| \|\vec{D}_{n-1,n}\|} & \text{if } n > 2 \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Then, U_n is used to determine the decision for the step size adaptation. The first case Figure 4(a) tends to produce negative dot products, whereas the second case tends to produce positive ones. Hence, when the value of U_n falls beyond a negative threshold, the step size of the agent will decrease, and when the value of U_n exceeds a positive threshold, the step size of the agent will increase (Algorithm 2).

Algorithm 2. Adaptation of the step size using the cumulative dot product

```

if  $U_n < -s_u$  then
     $R \leftarrow c_r R$ 
     $U_n \leftarrow 0$ 
else if  $U_n > s_u$  then
     $R \leftarrow \frac{1}{c_r} R$ 
     $U_n \leftarrow 0$ 
end
    
```

where c_r is constant in $]0,1]$, and s_u is a constant threshold, calculated as follows:

$$s_u = \cos\left(\frac{\pi}{3}\right) (1 + c_u) \quad (6)$$

This formula can be explained by the following considerations. In the case of $c_u = 0$, the value of U_n (see equation (5)) will be equal

to the cosine of the angle, denoted a , between the last two displacement vectors of an agent. As a can take any value in $[0, \pi]$, this interval has been uniformly partitioned and linked to the three following cases:

1. The agent follows a trajectory as in Figure 4(a). This case produces the highest values of a , and is associated to the interval $[\frac{2\pi}{3}, \pi]$;
2. The temporary situation, where we cannot take a decision about the agent. Hence, no step size adaptation is made in this case. It produces values of a around $\frac{\pi}{2}$, and it is related to the interval $[\frac{\pi}{3}, \frac{2\pi}{3}]$;
3. The agent follows a trajectory as in Figure 4(b). This leads to the lowest values of a , and this last case corresponds to the interval $[0, \frac{\pi}{3}]$.

This case ($c_u = 0$) corresponds to the case where the threshold value s_u is equal to $\cos(\frac{\pi}{3})$. Generally, c_u is not equal to 0, but can take any value in the interval $]0,1]$. We propose to adjust this threshold using equation (6).

The agents step size adaptation assumes that the agents are always able to move to a better position from their current one. This is a condition to be able to compute U_n from U_{n-1} and the last displacement. If an agent cannot find a better solution in its local landscape, the agent stays on its current position. Then, a second adaptation method is proposed in this particular case. This method is based on the following assumption: if the agent cannot find a better solution in its local landscape, then, it has probably finished hill climbing a peak, and an optimum may lay at a distance lower than the current radius of its neighborhood (the hypersphere of radius R). Then, reducing this

radius may let the agent continue its search towards the nearby optimum. Briefly, the second adaptation method consists in decreasing R in order to let the agent converge to the found optimum ($R \leftarrow c_r R$).

3.2.3 The Convergence Process and the Stagnation Criterion

We consider that an agent converged to a local optimum, when none of its last δ_t steps improved significantly the fitness value of its current solution (if $|f_n - f_{n-1}| \leq \delta_p$ for $n = 1, 2, \dots, \delta_t$ where f_n is the fitness of the solution found by the agent n trajectory steps ago). This is a well known stagnation criterion for stopping a local search. We use the parameter δ_t to prevent unnecessary fitness evaluations. An adaptation method for the number of neighbor solutions, that have to be evaluated in order to find the new position of an agent, is based on the following considerations. When an agent is close to a local optimum, it has a lower probability to get out of this local optimum than an agent that is still exploring the landscape. The agent will not be likely to escape from the basin of attraction (the region of the search space, around an optimum, that leads to trajectories towards this optimum) it has converged to, even if it performs a lot of fitness evaluations. However, for an agent that is still exploring the landscape, it is important to maintain a good trajectory, in order to hill climb towards a high quality solution. The convergence level of an agent can be expressed by the number of its last d_t steps that did not improve its fitness more than δ_p . The Algorithm 3 summarizes the procedure to find the next position of an agent.

Algorithm 3. The search for a better solution among the neighboring ones

$$v \leftarrow v_c$$

$$\vec{P}_{best} \leftarrow \vec{P}_c$$

```

foreach point  $\vec{P}_i^{rst} \in S'$  do
   $v_i \leftarrow \text{evaluate } \vec{P}_i^{rst}$ 
  if  $v_i > v$  then
     $v \leftarrow v_i$ 
     $\vec{P}_{best} \leftarrow \vec{P}_i^{rst}$ 
  if  $\text{rand}(\delta_t) < d_t$  then
    return  $(\vec{P}_{best}, v)$ 
  end
end
end
return  $(\vec{P}_{best}, v)$ 

```

where v_c is the current fitness of the agent, \vec{P}_c is its current position, S' is the set of all neighbor candidate solutions of the agent, \vec{P}_i^{rst} is a reflected scaled translated position of a given solution, $\text{rand}(\delta_t)$ generates a uniform random number in $[0, \delta_t]$, d_t is the number of steps that the agent has made without improving its fitness more than δ_p , and \vec{P}_{best} and v are the new solution of the agent in its search trajectory and its new fitness value, respectively.

When an agent has finally found a local optimum, the agent manager sends this optimum to the coordinator, that will transmit it to the memory manager. Then, the coordinator shows to the agent manager where this agent will be repositioned, in order to perform a new trajectory search. A new value is given to the step size (R) of this agent. The way the new step size value is calculated is described in subsection 3.4. One can remark that in a flat landscape, agents will not move from their initial position (their trajectory will consist in a succession of the same initial point), and stop their local search after δ_t steps. Thus, in a flat landscape, the local optima transmitted to the memory manager will be always the initial positions of the agents. In this particular case, the only adaptation of the step size of an agent is a decrease in R , since no displacement vectors can be computed (the agent does not move).

3.2.4 The Diversity Maintaining Strategy

To prevent several agents from exploring the same zone of the search space, and to prevent them from converging to the same local optimum, an exclusion radius r_e is attributed to each agent. Hence, when an agent detects one or several other agents at a distance lower than r_e , only the agent with the best fitness, among the detected agents and the agent having detected them, is allowed to continue its search. All the other agents have to start a new search elsewhere (see subsection 3.4 to see how they are repositioned). The value of r_e is dynamically adjusted each time a new local optimum is memorized, and its new value is calculated by the memory manager. To prevent that r_e becomes too small, a given radius r_l needs to be introduced. This radius will be used as a lower bound for r_e . It is required, in order to avoid the use of near zero values for r_e and as initial step size. Too small values for these radius will slow the adaptation processes. The update process of r_e is administrated by the coordinator, and executed by the memory manager.

3.2.5 The Flowchart of an Agent

Agents perform their search by running their local search algorithm independently of each other. The flowchart of the agents search algorithm is illustrated in Figure 5. One can remark that a special state named "SYNCHRONIZATION" appears two times in this flowchart. This state marks the end of one "loop" of the algorithm of an agent. Hence, when this state is reached, the agent manager halts the execution of this agent until all other agents have reached a SYNCHRONIZATION state. Then, the execution of the agents is resumed at the SYNCHRONIZATION state they previously halted on (not at another SYNCHRONIZATION state). This special state allows then the parallel execution of the agents.

3.3 Memory Management Module

3.3.1 Local Optima Archiving Strategy

The memory manager maintains the archive of local optima found by the agents. This archive must be bounded, its size is fixed by a predefined number n_m of entries. Thus, all the found optima cannot be included into this archive, only the n_m best ones will be stored. When the archive is full, we propose the following condition to update the archive:

- If the new optimum is better than the worst optimum of the archive, or its fitness value is at least equal to the one of this worst optimum, then this worst optimum is replaced by the new one;
- If there is one or several other optima in the archive that are "too close" to the new optimum, then it is possible that all these optima close to each other are in fact scattered around the top of one peak. Thus, this subset of solutions is replaced by the best optimum among them. Hence, if there are other optima in the archive that are "too close" to the newly found one, this new optimum will replace them, only if it is better, or if its fitness is at least equal to the fitness of the best one. The process of detecting possible "too close" optima is presented in Algorithm 4.

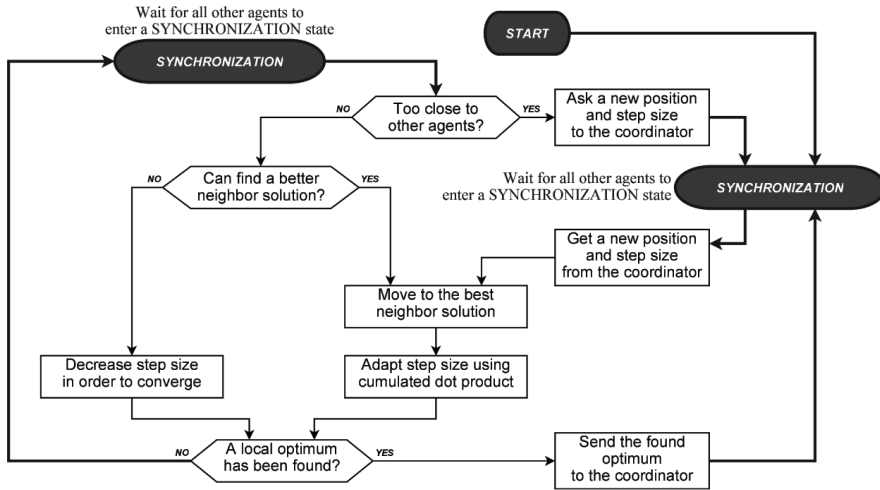
Algorithm 4. The replacement of suboptimal stored solutions

```

 $S_{sub} \leftarrow \{O_c\}$ 
 $O_{best} \leftarrow O_c$ 
foreach  $O_i \in S_m$  do
  if  $dist(O_i, O_c) \leq r_o$  then
     $S_{sub} \leftarrow S_{sub} \cup \{O_i\}$ 

```

Figure 5. The main algorithm flowchart of a MADO agent



if $fitness(O_i) > fitness(O_{best})$ *then*

$O_{best} \leftarrow O_i$

end

end

end

$S_{sub} \leftarrow S_{sub} - \{O_{best}\}$

$S_m \leftarrow S_m - S_{sub}$

$S_m \leftarrow S_m \cup \{O_{best}\}$

where O_c is the newly found optimum, S_m is the archive of stored optima, $dist(O_i, O_c)$ computes the distance between O_i and O_c , $fitness(O_i)$ is the fitness value of O_i and r_o is a threshold that defines the boundary of the suboptimal zone around an optimum, where there is not any other optimum.

r_o does not need to be lower than the lowest initial step size r_l of an agent, because even if there was another optimum O' at a distance from O_{best} lower than r_l , an agent starting a new search from O_{best} would detect O' . This is due to the fact that O' will lie in the local landscape of the agent (inside the hypersphere of radius R with $R \geq r_l$). Then, an agent starting

a new search from O_{best} would be able to sample candidate solutions around O' . Moreover, r_o must be high enough in order, for most stored suboptimal solutions, to be detected and removed. One can remark that, in the case where the landscape has thin peaks close to each other, a high value of r_o will remove some of them. The value of r_e is adjusted in order to prevent agents from hill climbing a same peak, and this value should be indeed a good upper bound for r_o . From these considerations, a good choice for the value of r_o appeared to be the geometric average of r_l and r_e . The value of r_o is thus calculated according to (7).

$$r_o = \sqrt{r_l r_e} \quad (7)$$

3.3.2 The Computation of the Exclusion Radius

Another task assigned to the memory manager is updating the value of r_e . At the beginning, an initial value of r_e , denoted by r_e^{max} , is calculated according to (8).

$$r_e^{max} = \frac{1}{2 n_a} \quad (8)$$

where n_a is the predefined number of agents created in the search space at the beginning of the MADDO algorithm. As it will be shown in subsection 3.4, the number of agents may vary temporarily, but the average number of agents along the whole search process tends to be equal to n_a . According to (8), when n_a increases, r_e^{max} decreases. We have to take into account that the increase in the number of agents will increase the probability to have several agents converging to the same optimum. Thus, if r_e^{max} does not decrease according to the number of agents, then for a high number of agents, most of them will never start a new local search, because of the presence of other agents in their exclusion radius. The formula (8) has also been chosen because it gives the largest radius that an agent can have. It allows to place all the n_a agents in a one-dimensional

search space, without having their exclusion radius overlapping another one.

When a new optimum is found, it is transmitted to the memory manager, and if this optimum is accepted, then the memory manager will update the value of r_e using this last stored optimum. The process of updating r_e is presented in Algorithm 5; where S'_m is the subset of optima in the archive without the last stored optimum, S_m is the set of stored optima, O_c is the last stored optimum, $dist(O_i, O_c)$ computes the distance between O_i and O_c and $card(S'_m)$ is the number of optima in the set S'_m . To update r_e , we compute the average value of the lowest distances from each stored optimum to the others, denoted by δ_{mean} . This is done using (9).

$$\delta_{mean} = \frac{\sum_{O_i \in S'_m} \left(\min_{O_j \in S_m - \{O_i\}} dist(O_j, O_i) \right)}{card(S'_m)} \quad (9)$$

Algorithm 5. The updating of the exclusion radius

```

 $S'_m \leftarrow S_m - \{O_c\}$ 
if  $card(S'_m) > 0$  then (since we need to compute distances, there must be at least two optima
in the archive)
     $\delta_{min} \leftarrow \infty$ 
    foreach  $O_i \in S'_m$  do (computes the lowest distance between the new optimum and the
    previously stored ones)
         $\delta_i \leftarrow dist(O_i, O_c)$ 
        if  $\delta_i < \delta_{min}$  then
             $\delta_{min} \leftarrow \delta_i$ 
        end
    end
    (since a too high  $r_e$  might prevent the convergence to optima close to each other,
    if  $\frac{\delta_{min}}{2} < r_e^{max}$  then  $r_e^{max}$  is used as an upper bound for  $\frac{\delta_{min}}{2}$ )
     $r_{mean} \leftarrow \frac{card(S'_m) r_e + \frac{\delta_{min}}{2}}{card(S'_m) + 1}$ 
     $r_e^{new} \leftarrow \frac{card(S'_m) r_{mean} + (n_m - card(S'_m)) r_e}{n_m}$ 
     $r_e \leftarrow r_e^{new}$ 
end
end
    
```

Assuming that r_e tends to be equal to the average value of the lowest distances from each optimum of S'_m to the others, then we can replace the sum of these lowest distances in (9) by $2 \text{card}(S'_m) r_e$. This leads to the formula (10).

$$\delta_{mean} \approx \frac{2 \text{card}(S_m - \{O_c\}) r_e + \min_{O_j \in S_m - \{O_c\}} \text{dist}(O_j, O_c)}{\text{card}(S_m)} \quad (10)$$

Since the exclusion radius r_e should match the average “radius” of the attractive zone of a peak, it is estimated by half the average lowest distance between the top of two peaks. This value, denoted by r_{mean} , is calculated from δ_{mean} using (11):

$$r_{mean} = \frac{\delta_{mean}}{2} \quad (11)$$

When S_m is nearly empty, the updating process does not have enough optima to estimate a fitted average lowest distance between peaks. Then, it should not modify the value of r_e so much. However, when S_m will be filled out, this estimation will be adapted to the search space landscape, and r_e will be made close to r_{mean} . This is the idea behind the expression of r_e^{new} in Algorithm 5.

3.4 Coordinator

The coordinator administrates all the main operations of the search process, by giving instructions to memory and agent management modules, and receiving information from them. It is in charge of the creation of the agents at the beginning of the algorithm. The number of agents to be created is given by the parameter n_a . The locations of these agents at the start of the search process are not randomly generated, but are computed in order to maximize

distances between them. This is done by using the electrostatic repulsion heuristic described in section 3.2.1. Rather than constraining the repulsing points to stay on the surface of the unit hypersphere, they are constrained to stay inside the hyperrectangle, denoted by T , that is centered in the search space (Figure 6). As illustrated in Figure 6, T is the Cartesian product of intervals $T_i = [r_e^{max}, 1 - r_e^{max}]$ with $i = 1, 2, \dots, d$. Thus, at the end of this heuristic, we get a set of initial positions for the initial set of agents, which are optimally covering the search space, i.e. the lowest distance between two of them is maximized inside T .

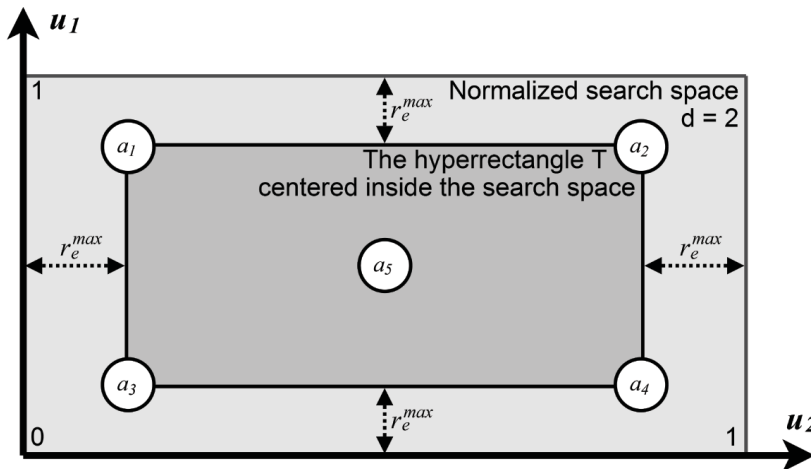
The instruction to create the initial set of agents is given to the agent management module, then the MADDO algorithm can start the search process. The exploration of the search space is done by the agents, using trajectory local searches. Once an agent has found an optimum, this optimum is transmitted to the coordinator, that transmits it to the memory management module. Afterwards, the coordinator transmits a new location and a new step size to the agent. This new location is randomly generated and uniformly distributed in the search space, under the constraint that the lowest distance between this new location and the current position of any agent is greater or equal to the exclusion radius r_e . After five attempts to generate a new location at a random position outside the exclusion radii of agents without success, the coordinator sends a delete instruction to the agent manager. We did so because of the high probability that the search space is saturated with existing agents. If a random position satisfying the constraint is found, it is given as the new position of the agent, and its new step size is calculated as in (12):

$$R = \max(r_e, \text{rand}(1)) \quad (12)$$

where $\text{rand}(1)$ generates a uniform random number in $[0, 1]$.

The coordinator detects also the changes in the environment. This detection is performed when all the agents have finished one loop in

Figure 6. Computation of an initial set of 5 agents in a two-dimensional search space



Agents are depicted by white-filled circles denoted by a_i with $i \in \{1, 2, \dots, 5\}$

their search algorithm, i.e. when all the agents have moved one step ahead in their trajectory and/or performed one adaptation of their step size.

Changes in the environment are detected by re-evaluating the fitness of the best optimum of the archive. When any optimum has been found yet, we re-evaluate the current fitness of an agent randomly chosen, and compare it to its previous value. If these values are different, a change is supposed to have occurred, and a tracking of the stored optima is performed (Algorithm 6); where S_a is the set of all agents, S_m is the

archive of stored optima and r_i is the lowest possible initial step size of an agent, i.e. the lowest allowed value for r_e . The use of such a low initial step size for these new agents is needed in order to track the optima, i.e. to hill climb the peaks of the previously stored optima to find their new position. A large initial step size allows these agents to leave the peaks of the tracked optima. Thus, they will explore other zones of the search space rather than staying on the peaks they have to hill climb. Hence, the best value for r_i is the value that takes

Algorithm 6. The tracking of stored optima after a change

```

foreach  $a_i \in S_a$  do
    Re-evaluate the fitness of  $a_i$ 
end
foreach  $O_i \in S_m$  do
    Re-evaluate the fitness of  $O_i$ 
    Create an agent located on  $O_i$  and with an initial step size of  $r_i$ 
end
 $S_m \leftarrow \emptyset$ 

```

into account the move of the peaks through the search space.

As we can see, the number of agents varies along the search process, and can be lower or greater than the initial number n_a . However, to prevent the number of agents to increase more and more at every detection of a change in the environment, the coordinator will send a delete instruction if the number of agents is higher than n_a , rather than letting an agent start a new search, when its trajectory search is finished. This way, the number of agents will be temporarily higher than n_a (after a change has been detected), and will decrease to n_a , because the excess agents will just track the previously stored optima and send back their new positions.

3.5 Parameter Setting of MADO

Table 2 summarizes the eight parameters of MADO that the user has to define. In this table, the values given are suitable for MPB and they were fixed experimentally. These values will be used to perform the experiments reported in Section 4.

We fixed the number of agents equal to 4, because the convergence needs to be fast. Having too many agents exploring the search space will slow down the individual convergence of each agent and lead to a waste of fitness function evaluations. The precision parameter δ_p and the lowest step size r_l need to be correctly adapted to the change severity and to the change frequency of the environment. A low value for δ_p makes the agents unable to track optima in a fast changing environment, since they will spend too many iterations on fine-tuning their current solution, and might never be able to send it to the coordinator for archiving, before a new change in the environment occurs. Hence, the lower is the number of iterations between two changes, the higher should be the value of δ_p . This means that the precision of the found optima in fast changing environments might be worse than in slow changing ones.

The value of r_l needs also to be well suited to the severity of the changes, since a low value of r_l in a fast changing environment requires a lot of adaptations of the step size, and a waste of many fitness function evaluations. On the opposite, a high value of r_l may make the tracking agent leave the peak of the optimum it has to track, and make it begin exploring the search space elsewhere. The other parameters of the algorithm are much less critical, and might be left at their default values (given in Table 2).

4. RESULTS AND DISCUSSION

We used the Moving Peaks Benchmark in order to test the MADO algorithm. In (Branke, 1999), three sets of parameters, called scenarios, were proposed. These scenarios are given as a set of standard settings, that can be used in order to produce comparable results. Authors have to specify the scenario they used to test their methods, preventing the use of custom settings that may differ from one paper to another. It appears that the most commonly used set of parameters for MPB is scenario 2 (see Table 3), hence, it will be used in this article. In this scenario, the correlation coefficient λ is equal to 0, which means that when a change occurs, the peaks move in random directions that are not correlated to their previous moves. Hence, the direction of the peaks' moves cannot be predicted from the direction of their previous moves. This leads to a harder scenario than if we used a value of λ strictly greater than 0. The rest of this section is organized as follows. Firstly, an experimental analysis of the MADO algorithm is made, justifying the use of each component of the algorithm. This analysis is followed by a comparison to other methods on scenario 2, for various numbers of peaks and dimensions.

4.1 Empirical Analysis of the MADO Algorithm

An evaluation of all the different components of the MADO algorithm is made, in order to

Table 2. MADO parameter setting

Name	Interval	Default value	Short description
n_a	N	4	initial (and average) number of agents
n_m	N	10	capacity of the archive of found optima
N	N	$round(0.28 d + 2)$	number of candidate solutions per agent's move, where $round(v)$ gives the nearest integer to v
c_u	$[0, 1]$	0.5	coefficient which defines the weight of the cumulative dot product in the step size adaptation process
c_r	$]0, 1]$	0.8	coefficient which specifies how much the step size is decreased (or increased) during its adaptation
r_l	$]0, 1]$	0.0025	lowest possible step size (should match the change severity of the search space)
δ_t	N	8	maximum number of moves an agent can make without improving its fitness more that δ_p
δ_p	R	0.005	the precision parameter of the stagnation criterion of the agents trajectory searches

justify their requirement for obtaining quality results. This evaluation is performed on scenario 2 of MPB, and the resulting offline errors and standard deviations averaged on 100 runs are summarized in Table 4, for different variants of MADO algorithm. The maximum number of iterations is fixed to $5 \cdot 10^5$, that corresponds to 100 changes occurred in the environment during each run.

As one can see, the standard MADO is better than all the simpler variants tested here. Among these variants, we can note that $MADO - r_e$ does not perform a detection of other agents inside the exclusion radius r_e of an agent, i.e. an agent has not to start a new search elsewhere when it is too close to another agent, and the coordinator has not to generate a new starting position, for an agent outside the exclusion radius of the other agents. Thus, all agents can explore the same zone of the search space. However, r_e is still used in $MADO - r_e$ for the other procedures, as in equation (12) and in Algorithm 4. In $MADO - r_e^{adapt}$, the only

difference with the standard version is that r_e is constant and equal to r_e^{max} , i.e. the Algorithm 5 is not performed. In $MADO - S_a^{max}$,

the electrostatic repulsion heuristic is not used to produce the initial set of agents, since their positions are randomly generated inside the search space and outside the exclusion radius of existing agents. In $MADO - S$, the electrostatic

repulsion heuristic is not used to produce the set S of candidate solutions of an agent, and the candidate solutions are randomly generated and uniformly distributed inside the hypersphere of radius R centered on the agent's current position. In $MADO - CDPA$,

the cumulative dot product adaptation of the step size is not used, and the only adaptation process used is the reduction of R (when no better candidate solution can be found in the local landscape of an agent). Finally, we can also note that in $MADO - r_l^{track}$, the initial step size of an agent,

created by the coordinator when a change is

Table 3. MPB parameters in scenario 2

Parameter	Scenario 2
Number of peaks	10
Dimensions	5
Peak heights	[30, 70]
Peak widths	[1, 12]
Change cycle	5000
Change severity	1
Height severity	7
Width severity	1
Correlation coefficient λ	0

detected in the environment (on the location of a previously found optimum), is generated using equation (12).

From Table 4, one can conclude that the most important components of the MADO algorithm, in order to obtain good results, are the archiving of found local optima, the use of maximally spaced candidate solutions on the surface of the local landscape hypersphere of agents, and the use of r_i as initial step size of the agents created (in place of stored optima, when a change is detected).

To give an idea about the computational cost of the MADO algorithm, the average time on 100 runs needed to compute $5 \cdot 10^5$ fitness function evaluations on the scenario 2 of MPB, using a 2.26 GHz Intel Core 2 Duo P8400 processor, is equal to 363 milliseconds, including the time of evaluation of the fitness function of MPB.

4.2 Comparison with Competing Methods

The comparison, on MPB, of MADO with the other leading optimization algorithms in dynamic environments is summarized in

Table 5. The offline errors and the standard deviations are given, and the algorithms are sorted from the best to the worst. Results are averaged on 100 runs or 50 runs of the tested algorithms, and the maximum number of fitness evaluations per run is fixed to $5 \cdot 10^5$, i.e. 100 changes per run.

Results gathered in Table 5 for competing algorithms are those given in the references listed in the first column. As we can see, MADO is the second best algorithm in this classification, and its confidence interval overlaps the confidence interval of the best rated algorithm. Although MADO is rated behind MMEO (Moser & Hendtlass, 2007) on scenario 2 of MPB, MMEO produces worse results than MADO on some other instances of MPB, i.e. those with a higher number of peaks or dimensions. An extended comparison, averaged on 100 runs with 100 changes per run, between MMEO and MADO, is given in Table 6. Scenario 2 is still used, with a modified value of one parameter in each column, and the offline error is also used, with its standard deviation.

This comparison shows that MADO is more robust and can be applied to highly multimodal DOPs and DOPs with high dimensionality.

Table 4. Offline error on MPB for each simplified variant of the MADO algorithm

Variant	Offline error	Short description
$MADO$	0.80 ± 0.19	the standard MADO algorithm
$MADO_{n_a=1}$	1.26 ± 0.48	with only one exploring agent
$MADO_{n_m=0}$	5.96 ± 0.37	without archiving local optima
$MADO_{r_e}$	1.23 ± 0.38	no exclusion radius for the agents
$MADO_{r_e^{adapt}}$	0.82 ± 0.24	no adaptation of the exclusion radius
$MADO_{S_a^{max}}$	0.84 ± 0.26	no lowest distance maximization for the initial set of agents
$MADO_S$	3.78 ± 0.48	no lowest distance maximization for candidate solutions
$MADO_{CDPA}$	1.06 ± 0.33	no cumulative dot product adaptation of the step size of the agents
$MADO_{R^{min}}$	0.91 ± 0.33	using r_l instead of r_e as lower bound of the initial step size of an agent (see (12))
$MADO_{r_l^{track}}$	2.82 ± 0.24	without using r_l as initial step size for agents created in place of stored optima

5. CONCLUSION

A new multiagent algorithm has been designed for continuous dynamic optimization problems. In this algorithm, a multiple trajectory local search strategy with an adaptive step size has been implemented. These local searches are conducted by the agents of the algorithm, and a memory was added for archiving the found optima. This way, when a change is detected in the environment, the archived optima can be re-used in order to track them through the search space. The agents and the memory are also encapsulated in two different modules, that are themselves coordinated by a third module. This third module, named the coordinator, takes all the global decisions and actions over the whole search process, like deleting or creating agents, and detecting changes in the

environment. This implementation has been specifically designed for dynamic continuous optimization, and proved its efficiency on the Moving Peaks Benchmark. This algorithm can also be expected to produce good results, when solving static objective functions.

The major drawback of MADO is that its critical parameters are not automatically adjusted along with the search process, and need to be fine-tuned by the user. In works in progress, we would like to make these critical parameters adaptive. We will also attempt to make this algorithm simpler and lighter. Some of its components may be subject to replacement or deletion, and the number of parameters may be reduced. Finally, as a lot of real world DOPs are multi-objective, or have a lot of constraints that can be dynamic, the proposed algorithm may also be modified to make it

Table 5. Comparison with competing algorithms on MPB (scenario 2)

Algorithm	number of runs	offline error \pm standard deviation
Moser & Hendtlass, 2007	100	0.66 \pm 0.20
MADO	100	0.80 \pm 0.19
	50	0.78 \pm 0.22
Lung & Dumitrescu, 2007	50	1.38 \pm 0.02
Lung & Dumitrescu, 2008	50	1.53 \pm 0.01
Blackwell & Branke, 2006	50	1.72 \pm 0.06
Mendes & Mohais, 2005	50	1.75 \pm 0.03
Li, Branke, & Blackwell, 2006	50	1.93 \pm 0.06
Blackwell & Branke, 2004	50	2.16 \pm 0.06
Du & Li, 2008	50	4.02 \pm 0.56

Table 6. Offline error of MADO and MMEO on several instances of MPB

Algorithm	on 100 peaks	in 7 dimensions
MADO	1.12 \pm 0.08	1.61 \pm 0.13
MMEO	1.47*	1.73 \pm 0.55

multi-objective and able to handle hard and dynamic constraints.

REFERENCES

- Blackwell, T., & Branke, J. (2004). Multi-swarm optimization in dynamic environments (LNCS 3005, pp. 489-500).
- Blackwell, T., & Branke, J. (2006). Multi-swarms, exclusion and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10, 459-472. doi:10.1109/TEVC.2005.857074
- Branke, J. (1999). Memory enhanced evolutionary algorithms for changing optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999)* (pp. 1875-1882). Washington, DC: IEEE Computer Society.
- Branke, J. (1999). *The moving peaks benchmark*. Retrieved from <http://www.aifb.uni-karlsruhe.de/~jbr/MovPeaks>
- Branke, J., & Mattfeld, D. (2005). Anticipation and flexibility in dynamic scheduling. *International Journal of Production Research*, 43, 3103-3129. doi:10.1080/00207540500077140
- Conway, J. H., & Alexander, N. J. (1998). *Sphere packings, lattices and groups* (3rd ed.). New York: Springer.
- Dorigo, M., & Gambardella, L. M. (2002). Guest editorial special on ant colony optimization. *IEEE Transactions on Evolutionary Computation*, 6, 317-319. doi:10.1109/TEVC.2002.802446
- Dréo, J., & Siarry, P. (2006). An ant colony algorithm aimed at dynamic continuous optimization. *Applied Mathematics and Computation*, 181, 457-467. doi:10.1016/j.amc.2005.12.051

- Du, W., & Li, B. (2008). Multi-strategy ensemble particle swarm optimization for dynamic optimization. *Information Sciences*, 178, 3096–3109. doi:10.1016/j.ins.2008.01.020
- Hansen, N., & Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9, 159–195. doi:10.1162/106365601750190398
- Householder, A. (1958). Unitary triangularization of a nonsymmetric matrix. *Journal of the ACM*, 5, 339–342. doi:10.1145/320941.320947
- Huang, C.-F., & Rocha, L. M. (2005). Tracking extrema in dynamic environments using a coevolutionary agent-based model of genotype edition. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation* (pp. 545-552). ACM Publishing.
- Jin, Y., & Branke, J. (2005). Evolutionary optimization in uncertain environments - a survey. *IEEE Transactions on Evolutionary Computation*, 9, 303–317. doi:10.1109/TEVC.2005.846356
- Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks* (pp. 1942-1948). Washington, DC: IEEE Computer Society.
- Larsen, A. (2000). *The dynamic vehicle routing problem*. Copenhagen, Denmark: Technical University of Denmark.
- Li, X., Branke, J., & Blackwell, T. (2006). Particle swarm with speciation and adaptation in a dynamic environment. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation* (pp. 51-58). ACM Publishing.
- Lung, R. I., & Dumitrescu, D. (2007). Collaborative evolutionary swarm optimization with a Gauss Chaotic Sequence Generator. *Innovations in Hybrid Intelligent Systems*, 44, 207–214. doi:10.1007/978-3-540-74972-1_28
- Lung, R. I., & Dumitrescu, D. (2008). ESCA: A new evolutionary-swarm cooperative algorithm. *Studies in Computational Intelligence*, 129, 105–114. doi:10.1007/978-3-540-78987-1_10
- Mendes, R., & Mohais, A. (2005). DynDE: A differential evolution for dynamic optimization problems. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (pp. 2808-2815). Washington, DC: IEEE Computer Society.
- Minner, S. (2003). Multiple-supplier inventory models in supply chain management: A review. *International Journal of Production Economics*, 81, 265–279. doi:10.1016/S0925-5273(02)00288-8
- Moser, I., & Hendtlass, T. (2007). A simple and efficient multi-component algorithm for solving dynamic function optimisation problems. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (pp. 252-259). Washington, DC: IEEE Computer Society.
- Nelder, J., & Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, 7, 308–313.
- Rossi, C., Abderrahim, M., & Diaz, J. C. (2008). Tracking moving optima using Kalman-based predictions. *Evolutionary Computation*, 16, 1–30. doi:10.1162/evco.2008.16.1.1
- Rossi, C., Barrientos, A., & Cerro, J. D. (2007). Two adaptive mutation operators for optima tracking in dynamic optimization problems with evolution strategies. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (pp. 697-704). ACM Publishing.
- Simoes, A., & Costa, E. (2008). Evolutionary algorithms for dynamic environments: Prediction using linear regression and Markov chains. In *Parallel problem solving from nature* (pp. 306-315). Springer.
- Tfaily, W., & Siarry, P. (2008). A new charged ant colony algorithm for continuous dynamic optimization. *Applied Mathematics and Computation*, 197, 604–613. doi:10.1016/j.amc.2007.08.087
- Tinos, R., & Yang, S. (2007). A self-organizing random immigrants genetic algorithm for dynamic optimization problems. *Genetic Programming and Evolvable Machines*, 8, 255–286. doi:10.1007/s10710-007-9024-z
- Yang, S. (2003). Non-stationary problem optimization using the primal-dual genetic algorithm. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation* (pp. 2246-2253). Washington, DC: IEEE Computer Society.
- Yang, S. (2006). A comparative study of immune system based genetic algorithms in dynamic environments. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation* (pp. 1377-1384). ACM Publishing.

Yang, S., & Yao, X. (2005). Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Computing - A Fusion of Foundations . Methodologies and Applications*, 9, 815–834.

Julien Lepagnot was born in France in 1982. He received the master's degree in computer sciences in 2008 from the University Evry-val-d'Essonne. Now, he is pursuing a PhD degree in computer engineering at the University Paris 12. His main research interests include image segmentation and compression, stochastic global optimization heuristics and multi-agent systems.

Amir Nakib was born in Algeria in 1977. He received the magister in signal processing in 2003 and a second MS degree in image processing in 2004 from the University Paris 6. He received in December 2007 a PhD degree in computer engineering at University Paris 12 (France). His main research interests include image segmentation and compression, signal compression, and stochastic global optimization heuristics.

Hamouche Oulhadj was born in Algeria in 1956. He received the BS degree in electrical engineering at the Polytechnic School of Algiers, the DEA and the PhD degree in biomedical engineering from the University Paris 12 in 1985 and 1990 respectively. Now, he is an associate professor at the same University. His main research interests are pattern recognition, biomedical image segmentation and information extraction.

Patrick Siarry was born in France in 1952. He received the PhD degree from the University Paris 6, in 1986 and the doctorate of sciences (Habilitation) from the University Paris 11, in 1994. He was first involved in the development of analog and digital models of nuclear power plants at Electricité de France (E.D.F.). Since 1995 he is a professor in automatics and informatics. His main research interests are computer-aided design of electronic circuits, and the applications of new stochastic global optimization heuristics to various engineering fields. He is also interested in the fitting of process models to experimental data, the learning of fuzzy rule bases, and of neural networks.