2009 Ninth International Conference on Intelligent Systems Design and Applications

Performance analysis of MADO dynamic optimization algorithm

Julien Lepagnot, Amir Nakib, Hamouche Oulhadj and Patrick Siarry
Laboratoire Images, Signaux et Systèmes Intelligents, LISSI, E.A. 3956
Université de Paris 12
61 avenue du Général de Gaulle, 94010 Créteil, France
Email: siarry@univ-paris12.fr

***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 paper, a new multiagent algorithm for solving dynamic problems is studied. This algorithm, called MADO, is analyzed using the Moving Peaks Benchmark, and its performances are compared to those of competing dynamic optimization algorithms on several instances of this benchmark. The obtained results show the efficiency of MADO, even in multimodal environments.**

***Keywords*-dynamic; non-stationary; time-varying; continuous; optimization; multiagent; metaheuristic; Moving Peaks.**

I. 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. For dynamic environments, the goal is not only to locate the optimum, but to follow it as closely as possible.

In this paper, a simplified and improved version of MADO algorithm is proposed. The algorithm, called "MADO" for "MultiAgent Dynamic Optimization", has been developed in order to solve a wide range of continuous DOPs efficiently. It is a multiagent based method, that makes use of a population of agents to explore the search space. The optima found by the agents are archived in a memory, to be used when a change is detected.

The rest of this paper is organized as follows. Section 2 discusses related works. Section 3 describes the proposed algorithm. Section 4 presents the benchmark set used to test the algorithm. Experimental results are discussed in Section 5. Conclusion and work under progress are evoked in section 6.

II. Related works

In this section, we present competing dynamic metaheuristics that have been proposed in the literature. We only focus on the different techniques for handling continuous DOPs. In principle, these techniques include restarting, multi-population, memory-based and diversity preserving (see [1]).

In [2], a multipopulation differential evolution (DE) algorithm is proposed, where 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 [2], these parameters are randomly generated in order to make the use of DE easier.

In [3], the authors propose two multi-swarms PSO algorithms based on an atomic model. The first algorithm uses several 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. This approach of using charged particles is also used in [4] and [5], with other techniques to increase diversity and to track optima.

Another PSO algorithm is proposed in [6], that uses two populations of particles. The first one is for the diversification, and the second is for the intensification. In [7] and [8], the authors make collaborate two kinds of populations: PSO swarm for the intensification, coupled with one or two other populations that follow the rules of evolutionary algorithms (EAs). These populations, based on EAs, are used to maintain diversity and to keep track of previously found solutions.

In [9], an algorithm based on extremal optimization (EO) is proposed. 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 the memory, and the method is applied again on another randomly generated solution. This algorithm is simple, efficient, and allows to obtain good results on a specific test called the "Moving Peaks Benchmark" (MPB) [10]. This method is especially developed and fitted for this benchmark.

978-0-7695-3872-3/09 $26.00 © 2009 IEEE
DOI 10.1109/ISDA.2009.35

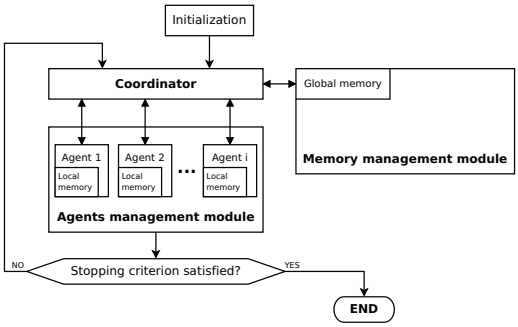


Figure 1. Overall scheme of MADO algorithm.

III. The proposed MADO algorithm

In this paper, we will only give an overview of the algorithm. For more details, see [11].

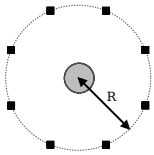
A. Overall scheme

MADO is a multiagent algorithm that consists in three modules: the agents manager, the memory manager and the coordinator. The overall scheme of the MADO algorithm is illustrated in figure 1. As it is shown, all the interactions between the memory manager and the agents 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 agents manager informs the coordinator about the found optima, and receives its instructions for creating, deleting, and repositioning agents.

As we consider 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, we use the modified unit vectors u_i, with $\{\vec{u}_1 = \frac{\vec{e}_1}{\Delta_1}, \vec{u}_2 = \frac{\vec{e}_2}{\Delta_2}, ..., \vec{u}_d = \frac{\vec{e}_d}{\Delta_d}\}$ (e_i form the canonical basis), as the basis of the Euclidian space where the search space is defined.

B. Agents management module

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. 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. We define the neighborhood of an agent as a set of N candidate solutions placed on the hypersphere of radius R centered on the current position of the agent. These candidate solutions are placed in a way that maximizes the smallest distance between them. This is done by initializing a set of N uniformly spaced points on the unit hypersphere (at the beginning of MADO) using a well known electrostatic repulsion heuristic described in [12]. Figure 2 illustrates this neighborhood in dimension 2.



The agent is depicted by a grey-filled circle. Candidate solutions are represented by black squares.

Figure 2. The sampling of candidate solutions of an agent ($d = 2, N = 8$).

Figure 3. 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.

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 [13], with much simpler calculations. The figure 3 illustrates the two possible kinds of "bad" trajectories that an agent may follow. When one of these cases is detected, a decrease (a) or an increase (b) of the step size R is performed, by a constant coefficient denoted by c_r. If the agent can not find a better solution in its neighborhood, R is decreased ($R \leftarrow c_r\ R$).

3) The diversity maintaining strategy: to prevent several agents from exploring the same zone of the search space, 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 having the best fitness is allowed to continue its search. All the other agents have to start a new search elsewhere [11].

In this new version of MADO, when the stepsize of an agent decreases below r_l, the algorithm in figure 4 is executed to prevent the agent from wasting fitness evaluations in order to converge to an already stored optimum.

4) 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 ($> \delta_p$). This is a well known stagnation criterion for stopping a local search. The parameter δ_t is also used to prevent the waste of unecessary fitness evaluations, when an agent explores its neighborhood. When an agent has finally found a local optimum, the agents manager sends this optimum to the coordinator, that will transmit it to the memory manager. Then, the coordinator shows to the agents manager where this agent will be repositioned, in order to perform a new trajectory search. In this new version of MADO, when an agent starts a new trajectory search, its step size (R) is reset to r_e, and r_e

$i \leftarrow 1$
while $i \leq card(S_m)$ **do**
  if $dist_a(O_i) \leq \sqrt{r_l \; r_e}$ and $f_i > f_c$ **then**
    Ask the coordinator to set a new position and step size for the agent
    $i \leftarrow card(S_m)$
  end
  $i \leftarrow i + 1$
end

Figure 4. Algorithm to prevent converging to a known optimum, where S_m is the archive of stored optima, $card(S_m)$ is the number of optima stored in S_m, O_i is the i^{th} optimum in S_m, $dist_a(O_i)$ is the distance between the agent and O_i, $\sqrt{r_l \; r_e}$ is the geometric average of the set $\{r_l, r_e\}$, f_i is the fitness of O_i, and f_c is the current fitness of the agent. In this algorithm, we suppose that we are in the case of a maximization problem. For a minimization problem, the condition $f_i > f_c$ becomes $f_i < f_c$.

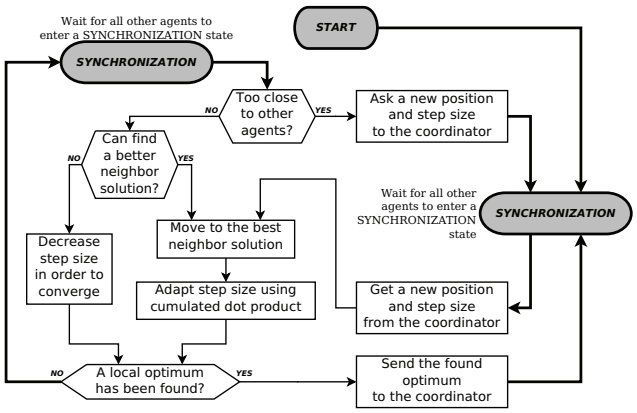


Figure 5. The main algorithm flowchart of a MADO agent.

remains constant throughout the run of MADO.

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 see 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 agents 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, where they previously halted on. This special state allows to run in parallel the agents.

C. Memory management module

1) The storing strategy of found local optima: 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, the following conditions are used to update the archive:

- if the fitness of the new optimum is better or equal to the worst optimum of the archive, 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 this subset of solutions is replaced by the best optimum among this set.

D. Coordinator

The coordinator administrates all the main operations of the search process, by giving instructions to the memory and agents 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 the lowest distance between two of them, inside an hyperrectangle denoted by T. (T is the Cartesian product of intervals $T_i = [r_e, 1 - r_e]$ with $i = 1, 2, ..., d$).

The instruction to create the initial set of agents is given to the agents management module, then the MADO algorithm can start the search process. In this new version of MADO, the initial step size of the agents is set to r_e.

The coordinator detects also the changes in the environment, by re-evaluating the fitness of the best optimum of the archive, and, if the archive is empty, the 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.

E. Parameter fitting of MADO

Table I summarizes the parameters of MADO that the user has to define. In this table, the values of the parameters are suitable for MPB and they were fixed empirically. These values will be used to perform the experiments reported in section V.

IV. Benchmark set

The most commonly used testbed for dynamic optimization is the Moving Peaks Benchmark (MPB) [10]. This benchmark is becoming the standard for testing dynamic optimization algorithms, and is claimed to be representative of real world problems [14]. To test and 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 can become 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 λ, $0 \leq \lambda \leq 1$, where 0 means uncorrelated and

Table I
MADO PARAMETER TYPICAL SETTING.

Name	Default	Short description
n_a	4	initial (and average) number of agents
n_m	10	capacity of the archive of found optima (depends on the degree of multimodality of the landscape: when there is a lot of local optima, higher values may produce better results)
N	3	number of neighbor candidate solutions (should grow with the number of dimensions, and depends on how fast the changes occur: in static environments, higher values should be used)
c_u	0.5	weight of the cumulative dot product (should be in $[0, 1]$)
c_r	0.8	step size adaptation coefficient (should be in $]0, 1]$, and depends on the ruggedness of the landscape: for a high ruggedness, c_r tends toward 1, and for a low one, c_r tends toward 0)
r_e	0.2	initial step size and exclusion radius (should be in $]0, 1]$, and depends on the ruggedness in the same way as c_r)
r_l	0.007	initial step size of tracking agents (should be in $]0, 1]$, and depends on the ruggedness in the same way as c_r, but also depends on the distance an optimum can move when a change occurs in the environment)
δ_t	2	maximum number of moves an agent can make without improving its fitness more than δ_p (depends on the ruggedness of the landscape: its value should grow with the ruggedness)
δ_p	0.001	precision parameter of the stagnation criterion of the agents trajectory searches (depends on the ruggedness in the same way as δ_t, but also depends on how fast the changes occur: in fast changing environments, we cannot spend a lot of time fine tuning the solution, thus δ_p should be higher)

1 means highly autocorrelated. The peaks change position every α iterations, and α is called time span.

In order to evaluate the performance, the "offline error" is used. The offline error (oe) is defined by:

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

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

We can see 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 [9].

In [10], three sets of parameters, called scenarios, were proposed. It appears that the most commonly used set of parameters for MPB is scenario 2 (see table II), hence, it will be used in this paper.

Table II
MPB PARAMETERS IN SCENARIO 2.

Parameter	Scenario 2
Num. 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

Table III
COMPARISON OF THE VERSIONS OF MADO ON MPB (SCENARIO 2).

Algorithm	offline error
New version of MADO	0.58 ± 0.10
Old version of MADO	0.68 ± 0.17

V. RESULTS AND DISCUSSION

A. Analysis of the parameters of MADO

In first, we compare the obtained results on MPB (scenario 2) of this new version of MADO and the old one. The algorithms are stopped after $5 \ . \ 10^5$ evaluations, and results are averaged on 100 runs. We used the parameter values in table I for both versions. The obtained results are summarized in table III. One can remark that the new version obtains better results with a lower standard deviation than that of the old one. In terms of complexity, this new version is less complex than the old one.

The robustness of MADO is studied by varying the value of one of its parameters, while the others are left fixed to their default values (see table I). Each parameter is studied this way, by applying the algorithm on MPB (scenario 2) and on the Rosenbrock function on five dimensions. The Rosenbrock function is used as a static minimization problem, that admits only one global optimum equal to 0. The search space used for Rosenbrock is $[-10, 10]^5$. The algorithm is stopped when $5 \ . \ 10^5$ evaluations have been performed on MPB, and when $4 \ . \ 10^4$ evaluations have been performed on Rosenbrock. The performance of MADO is measured on MPB using the offline error, and it is measured on Rosenbrock using the fitness of the best found solution. Results are averaged on 100 runs, and illustrated in figure 6 and figure 7.

As one can see, varying the value of n_m does not perturb the performance of MADO significantly on Rosenbrock. However, on MPB, the performance of MADO increases with the value of n_m. Thus, n_m is only significant in dynamic environment, and it has to be high enough to let the algorithm store a sufficient number of local optima. It appears that a value greater or equal to 10 is good for MPB. The best value for the parameter N (size of the neighborhood of an agent) on MPB and Rosenbrock is 3. However, in static environment, or for a higher number of dimensions, a higher value can produce better results. The

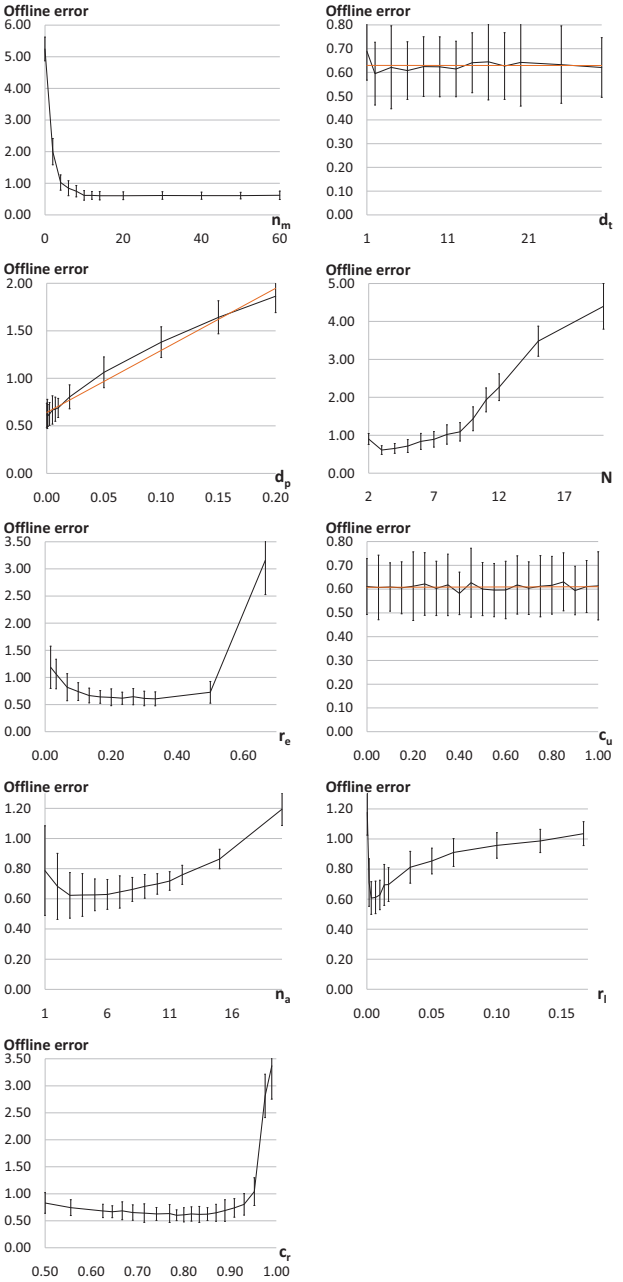


Figure 6. Robustness on MPB.

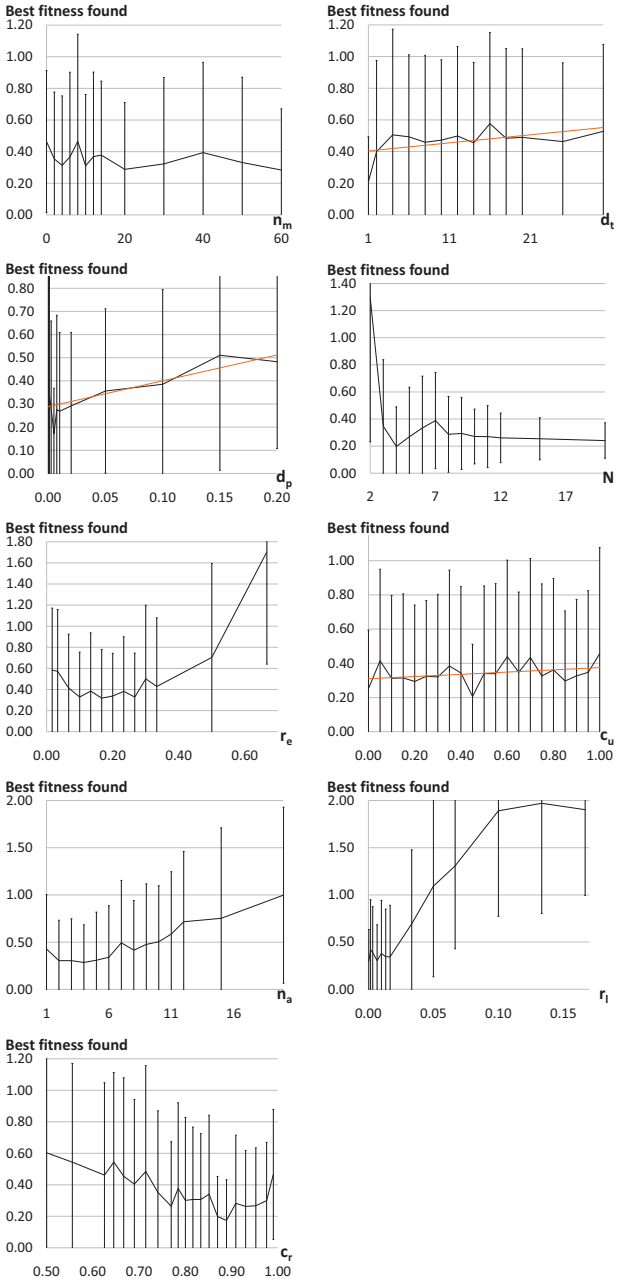


Figure 7. Robustness on Rosenbrock function.

behavior of n_a, d_t and c_u is similar on both test functions, and it appears that the best values of those parameters are $n_a = 4, d_t = 2, c_u = 0.5$. The best value of r_l in both cases is 0.007. However, good results can also be obtained in static environment with a lower value of r_l. The behavior of r_e, d_p and c_r is similar in both test cases, but the optimal values of these parameters appear to be problem dependent. Thus, even if their optimal values stay close to the default values, it is needed to fit these three parameters correctly.

The convergence of MADO is studied in figure 8, using MPB with scenario 2 in dimension 2. In this graph, the axis y corresponds to the first 10 time spans, the axis x corresponds to the first 1000 function evaluations of a time span with a granularity of 50 evaluations, and the axis z is equal to the relative error $\frac{f^*_y - f^*_{yX}}{f^*_y}$, where $X = 50\ x$, using the notations of equation (1). For more clarity, we used a logarithmic scale for z axis.

As one can see, the convergence of MADO in each time span is very fast. The first time span shows the highest values at the beginning, because MADO has not yet recorded the

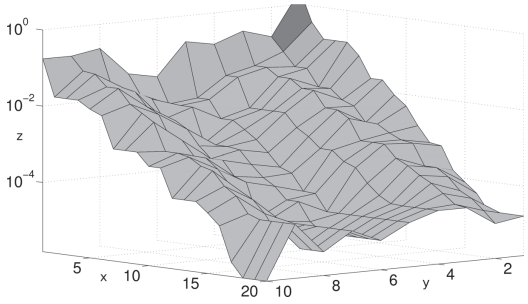


Figure 8. Relative error of the fitness on MPB.

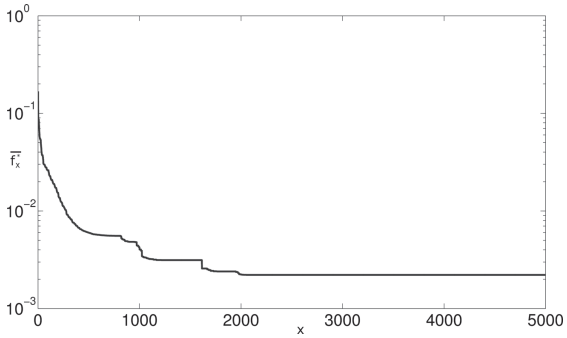


Figure 9. Average relative error of the fitness on MPB.

locations of each local optimum of the landscape. When the local optima are found, the algorithm converges faster by tracking them, rather than redetecting them. The average evolution of the relative error of the fitness among all time spans is given in figure 9, where x corresponds to the 5000 iterations of a time span, and $\bar{f}_x^*$ is the average value of f_x^* on 100 time spans. As illustrated in this figure, it takes only 3 iterations to get a relative error lower than 10^{-1}, and 278 iterations to get a relative error lower than 10^{-2}.

B. Comparison with competing methods

The comparison, on MPB, of MADO with the other leading optimization algorithms in dynamic environments is summarized in table IV. 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 $5 \,.\, 10^5$, *i.e.* 100 changes per run.

Results of competing algorithms are given in the references listed in the first column. As we can see, MADO allows to have better results than the other competing algorithms.

VI. Conclusion

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. However, MADO has four parameters that have to be correctly fitted, N, r_e, d_p and c_r. Another one, r_l, may also require some attention, though it is not as critical as the first four ones. The other three parameters of the algorithm should be left at their default values, since they do not perturb the performance significantly. Further studies should be made on the parameter N, to find a more widely applicable default value. Future work should be done to study co-dependencies between the parameters, and to make them auto-adaptive.

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

Algorithm	number of runs	offline error
MADO	100	0.58 ± 0.10
	50	0.59 ± 0.10
Moser and Hendtlass, 2007 [9]	100	0.66 ± 0.20
Lung and Dumitrescu, 2007 [7]	50	1.38 ± 0.02
Lung and Dumitrescu, 2008 [8]	50	1.53 ± 0.01
Blackwell and Branke, 2006 [4]	50	1.72 ± 0.06
Mendes and Mohais, 2005 [2]	50	1.75 ± 0.03
Li et al., 2006 [5]	50	1.93 ± 0.06
Blackwell and Branke, 2004 [3]	50	2.16 ± 0.06
Du and Li, 2008 [6]	50	4.02 ± 0.56

References

[1] Y. Jin *et al.*, "Evolutionary optimization in uncertain environments - a survey," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, 2005.

[2] R. Mendes *et al.*, "DynDE: A differential evolution for dynamic optimization problems," in *Congress on Evolutionary Computation*. Edinburgh, Scotland: IEEE, 2005, pp. 2808–2815.

[3] T. Blackwell *et al.*, "Multi-swarm optimization in dynamic environments," *Lecture Notes in Computer Science*, vol. 3005, pp. 489–500, 2004.

[4] T. Blackwell *et al.*, "Multi-swarms, exclusion and anti-convergence in dynamic environments," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 459–472, 2006.

[5] X. Li *et al.*, "Particle swarm with speciation and adaptation in a dynamic environment," in *Conference on Genetic and Evolutionary Computation*. Seattle, Washington, USA: ACM, 2006, pp. 51–58.

[6] W. Du *et al.*, "Multi-strategy ensemble particle swarm optimization for dynamic optimization," *Information Sciences*, vol. 178, no. 15, pp. 3096–3109, 2008.

[7] R. I. Lung *et al.*, "Collaborative evolutionary swarm optimization with a Gauss chaotic sequence generator," *Innovations in Hybrid Intelligent Systems*, vol. 44, pp. 207–214, 2007.

[8] R. I. Lung *et al.*, "ESCA: A new evolutionary-swarm cooperative algorithm," *Studies in Computational Intelligence*, vol. 129, pp. 105–114, 2008.

[9] I. Moser *et al.*, "A simple and efficient multi-component algorithm for solving dynamic function optimization problems," in *Congress on Evolutionary Computation*. Singapore: IEEE, 2007, pp. 252–259.

[10] J. Branke, "The moving peaks benchmark," available at http://www.aifb.uni-karlsruhe.de/∼jbr/MovPeaks, 1999.

[11] J. Lepagnot *et al.*, "A new multiagent algorithm for dynamic continuous optimization," to appear in International Journal of Applied Metaheuristic Computing (2010).

[12] J. H. Conway *et al.*, *Sphere Packings, Lattices and Groups*, 3rd ed. Springer, 1998.

[13] N. Hansen *et al.*, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[14] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *Congress on Evolutionary Computation*. Washington DC, USA: IEEE, 1999, pp. 1875–1882.