

A Dynamic Multi-Agent Algorithm Applied to Challenging Benchmark Problems

Julien Lepagnot, Amir Nakib, Hamouche Oulhadj, and Patrick Siarry
 Laboratoire Images, Signaux et Systèmes Intelligents, LISSI, E.A. 3956
 Université de Paris-Est Créteil
 61 avenue du Général de Gaulle, 94010 Créteil, France
 E-mail: {julien.lepagnot, nakib, oulhadj, siarry}@u-pec.fr

Abstract—Many real-world optimization problems are dynamic (time dependent) and require an algorithm that is able to continuously track a changing optimum over time. In this paper, we investigate a recently proposed algorithm for dynamic continuous optimization, called MLSDO (*Multiple Local Search algorithm for Dynamic Optimization*). MLSDO is based on several coordinated local search agents and on the archiving of the optima found over time. This archive is used when a change occurs in the objective function. The performance of the algorithm is evaluated on the set of benchmark functions provided for the IEEE WCCI-2012 Competition on Evolutionary Computation for Dynamic Optimization Problems.

I. INTRODUCTION

Recently, dynamic optimization 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 DOPs, the goal is not only to locate the global optimum, but also to follow it as closely as possible. Almost all algorithms for DOPs are population-based metaheuristics. We can roughly classify them in five categories: evolutionary algorithms (EAs), particle swarm optimization (PSO), ant colony optimization (ACO), artificial immune systems (AIS) and other algorithms (hybrid, multi-agent algorithms, local search based approaches).

The main goal of this paper is to evaluate the performance of our recently proposed dynamic optimization algorithm, called MLSDO [1], [2]. The main components of MLSDO are an evolving population of well diversified solutions, and an archiving of good solutions found during the search. More precisely, it makes use of a population of coordinated local searches to explore the search space. The use of local searches provides a fast convergence to the local optima, and the strategies used to coordinate these local searches enable the algorithm to widely explore the search space. The local optima found during the optimization process are archived, in order to be used when a change is detected. MLSDO is based on a multi-agent architecture [3].

The rest of this paper is organized as follows. Section II describes the fundamentals of the proposed MLSDO algorithm. Section III explains each strategy used in MLSDO. Section IV presents the benchmark sets used to test the algorithm. Experimental results are discussed in Section V. Conclusions and work under progress are presented in section VI.

II. THE MLSDO ALGORITHM

In the following subsections, we first describe how the distances are computed in the search space. Then, we describe the overall scheme of the proposed MLSDO algorithm and the initialization procedure.

A. Distance handling

In this paper, we propose to define the search space as a d -dimensional Euclidean space, since it is the simplest and most commonly used space. The inner product is given by the usual dot product, denoted by $\langle \cdot, \cdot \rangle$, and the Euclidean norm is denoted by $\|\cdot\|$.

Then, as the search space may not have the same bounds on each dimension, we use a “normalized” basis. We denote by Δ_i the size of each interval that defines the search space, where $i \in \{1, \dots, d\}$. 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 basis vectors (\vec{u}_i), defined as $\{\vec{u}_1 = \Delta_1 \vec{e}_1, \vec{u}_2 = \Delta_2 \vec{e}_2, \dots, \vec{u}_d = \Delta_d \vec{e}_d\}$. This change in basis transforms a hyper-rectangular search space into a hyper-square search space, according to (1), where x'_i and x_i are the i^{th} coordinates of a solution expressed in the hyper-square space, and in the hyper-rectangular space, respectively.

$$x'_i = \frac{x_i}{\Delta_i} \text{ for } i = 1, 2, \dots, d \quad (1)$$

B. Overall scheme

MLSDO is a multi-agent algorithm, that makes use of a population of agents to explore the search space. Agents are “nearsighted”: they only have a local view of the search space. More precisely, agents are only performing local searches; 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. A selection of these optima are saved in order to accelerate the convergence of the algorithm. The overall scheme of MLSDO consists of the following two modules (Figure 1):

- 1) **Memory manager:** in case of a multimodal objective function, a dynamic optimization algorithm needs to keep track of each local optimum found, since one of them can become the new global optimum after a change occurs in the objective function. Thus, we propose to

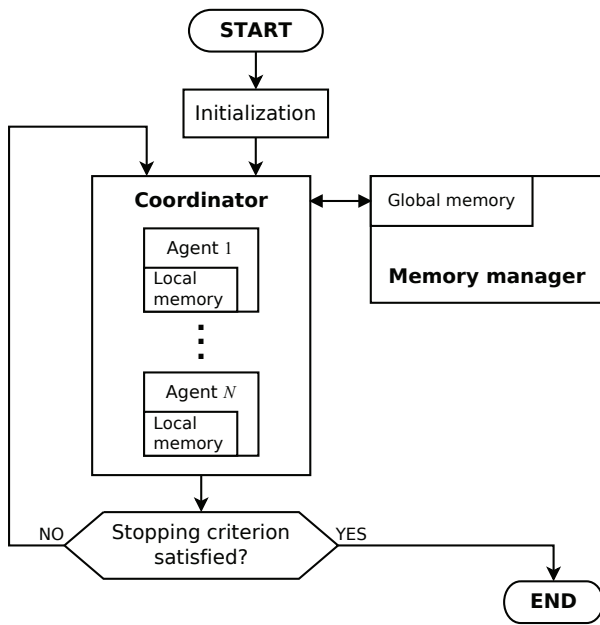


Fig. 1. Overall scheme of MLSDO. N is the number of currently existing agents. Each agent performs a search procedure described in subsection II-D.

save the found optima in memory. The memory manager maintains the archive of local optima that are provided by the coordinator.

- 2) **Coordinator:** it supervises the whole search, and manages the interactions between the memory manager and the agents. It compensates for the nearsightedness of the agents, and it is able to prevent them from searching in unpromising zones of the search space. The coordinator is informed about the found local optima, and manages the creation, destruction and relocation of the agents.

The initialization step in Figure 1 is discussed in subsection II-C and in section III. The stopping criterion depends on the optimization problem.

C. Computation of the initial set of agents

The coordinator starts by creating the agents in the initialization phase of MLSDO. The number of agents to be created is fixed by the parameter n_a . The initial positions of these agents are not randomly generated, but are computed in order to prevent several agents from being placed close to each other. This is done by sequentially placing the agents at the locations generated by a heuristic. This heuristic is detailed in subsection III-C.

Thus, at the end of this heuristic, we get a set of initial positions for the set of agents that is widely covering the search space.

D. The flowchart of an agent

Agents proceed by running their local search independently of each other. The flowchart of the search procedure of an agent is illustrated in Figure 2. One can see that two special states, named “SYNCHRONIZATION A” and

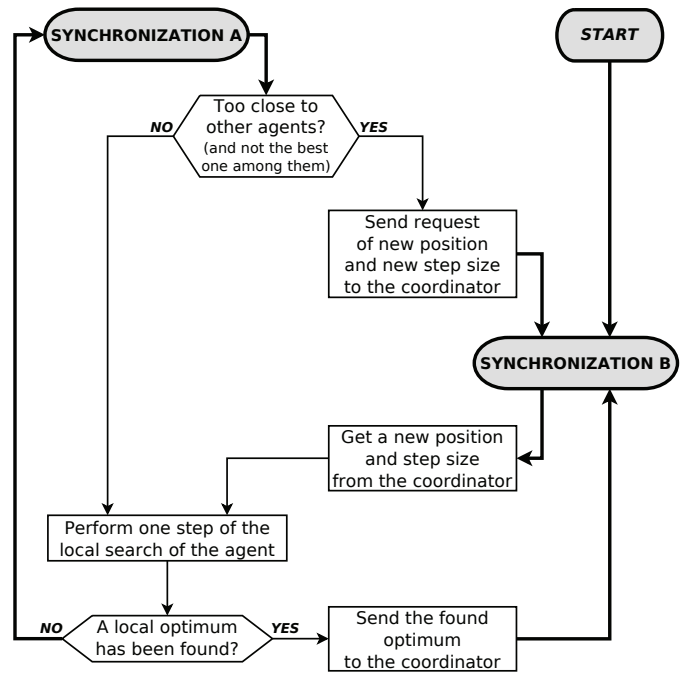


Fig. 2. Flowchart of the main procedure of a MLSDO agent.

“SYNCHRONIZATION B”, appear in this flowchart. These states mark the end of one step of the procedure of an agent. Hence, if one of these states has been reached, then the agent halts its execution until all other agents have reached one of these states. Afterwards, the execution of the agents is resumed; *i.e.*, if an agent halts on SYNCHRONIZATION A (SYNCHRONIZATION B, respectively), then it resumes its execution on SYNCHRONIZATION A (SYNCHRONIZATION B, respectively). This special state allows the parallel execution of the agents.

III. THE OPTIMIZATION STRATEGIES USED IN MLSDO

In the following subsections, the strategies used in MLSDO are described.

A. The exploration strategy of the agents

An MLSDO agent explores the search space step-by-step, moving from their current solution \vec{S}_c to a better one \vec{S}'_c in its neighborhood, until it reaches a local optimum. These step-by-step displacements are performed according to a step size R , adapted during the local search of the agent. An agent can be created for two reasons: to explore the search space or to track an archived optimum, when a change in the objective function is detected. In the first case, the agent is called an “exploring agent”, and in the second case, it is called a “tracking agent”. These two kinds of agents only differ in their initialization:

- **Exploring agents:** At the initialization of an exploring agent, its current solution \vec{S}_c is provided by the coordinator. The initial value of its step size R is equal to r_e , a parameter of MLSDO to be fixed.

localSearch

Inputs: $\vec{S}_c, d, R, U, r_l, \delta_{ph}, \delta_{pl}, \vec{D}$
local variables: $\vec{S}'_c, \vec{S}_w, stop, \vec{D}'$
 $\{\vec{S}'_c, \vec{S}_w\} \leftarrow \text{selectCandidate}(\vec{S}_c, d, R)$
 $stop \leftarrow \text{stoppingCriterion}(\vec{S}'_c, \vec{S}_w, \vec{S}_c, R, r_l, \delta_{ph}, \delta_{pl})$
Calculate the direction vector \vec{D}'
Update the cumulative dot product U
Update the step size R
if $\vec{S}'_c \neq \vec{S}_c$ **then**
 $\vec{S}_c \leftarrow \vec{S}'_c$
 $\vec{D} \leftarrow \vec{D}'$
end
if $stop = true$ **then**
 Stopping criterion (of the local search) satisfied: \vec{S}_c is the local optimum found
end
return $\{\vec{S}_c, R, U, \vec{D}\}$

Fig. 3. Procedure that performs one step of the local search of an agent. \vec{S}_c is the current solution of the agent. d is the dimension of the search space. R is the step size of the agent. U is the value of the *cumulative dot product* (see equation 2). r_l , δ_{ph} and δ_{pl} are parameters of the algorithm. \vec{D} is the direction vector of the last displacement of the agent. \vec{D}' is the direction vector of the displacement currently performed (see equation 3). The procedures *selectCandidate* and *stoppingCriterion* are described in subsection III-A.

- **Tracking agents:** Their current solutions are initialized using the best solutions of the archive containing the found local optima. The initial value of R is equal to r_l , a parameter of MLSDO that has to be lower than r_e . Indeed, an agent created to explore the search space requires a greater initial step size than a tracking agent.

At the initialization of the local search of an agent, in addition to \vec{S}_c and R , two other variables are initialized: the direction vector \vec{D} of the last displacement of the agent, and the *cumulative dot product* U (see equation 2), set to the null vector and zero, respectively. These variables are used to adapt R , as described later in this section.

We focus now on the local search of an agent, summarized in Figure 3 and described in detail below. The procedure in Figure 3 provides the details of the state indicated in Figure 2 by the description “perform one step of the local search of the agent”. Thus, it is repeated each time the agent is in this “local search” state, so as to enable the convergence of the local search. This is the main procedure of the local search performed by an agent, and it calls the subprocedures *selectCandidate* and *stoppingCriterion* that are defined below.

At each step of its local search, the agent moves from its current solution \vec{S}_c to the new candidate solution \vec{S}'_c according to the mechanism in Figure 4. As we can see, two candidate solutions are evaluated per dimension of the

selectCandidate

Inputs: \vec{S}_c, d, R
local variables: $i, \vec{S}_{prev}, \vec{S}_{next}, \vec{S}_i$
 $\vec{S}'_c \leftarrow \vec{S}_c$
 $\vec{S}_w \leftarrow \vec{S}_c$
for $i = 1$ **to** d **do**
 $\vec{S}_{prev} \leftarrow \vec{S}'_c - R \times \vec{u}_i$
 $\vec{S}_{next} \leftarrow \vec{S}'_c + R \times \vec{u}_i$
 Repair \vec{S}_{prev} if it is outside the search space
 Repair \vec{S}_{next} if it is outside the search space
 Evaluate \vec{S}_{prev} and \vec{S}_{next}
 $\vec{S}_i \leftarrow$ the best solution among \vec{S}_{prev} and \vec{S}_{next}
 if \vec{S}_i is strictly better than \vec{S}'_c **then**
 $\vec{S}'_c \leftarrow \vec{S}_i$
 end
 $\vec{S}_i \leftarrow$ the worst solution among \vec{S}_{prev} and \vec{S}_{next}
 if \vec{S}_i is worse or equal to \vec{S}_w **then**
 $\vec{S}_w \leftarrow \vec{S}_i$
 end
end
return $\{\vec{S}'_c, \vec{S}_w\}$

Fig. 4. Selection mechanism: selects the candidate solution \vec{S}'_c to replace the current solution \vec{S}_c of an agent. d is the dimension of the search space. R is the step size of the agent. \vec{u}_i is the basis vector of the i^{th} axis of the search space. \vec{S}_w is the worst tested candidate solution. The function *Evaluate* computes the value of the objective function of a given solution and assigns this value to the solution.

search space, denoted by \vec{S}_{prev} and \vec{S}_{next} . They stand in opposite directions from \vec{S}'_c along an axis of the search space, at equal distance R from \vec{S}'_c . If \vec{S}_{prev} or \vec{S}_{next} is outside the search space, it is “repaired” by setting it to the closest point inside the search space. For each axis of the search space, the best solution among \vec{S}_{prev} , \vec{S}_{next} and \vec{S}'_c becomes the new candidate solution \vec{S}'_c .

At the end of this procedure, the worst candidate solution \vec{S}_w is also returned, because it is used in the stopping criterion of the agent.

The adaptation of the step size R is based on the *cumulative dot product* U , that makes use of trajectory information gathered along the steps of the agent (using the successive direction vectors of the displacements of the agent). Thus, at each iteration of the local search of an agent, the value of U is firstly updated, then R is adapted according to the new value of U . This new value, denoted by U' , is computed according to the equation (2), where $\langle \vec{D}, \vec{D}' \rangle$ is the dot product of the last two direction vectors (the direction vectors of the previous and the current steps of the agent, respectively). The direction vector \vec{D}' (from \vec{S}_c to \vec{S}'_c) is calculated according to the equation (3).

$$U' = \frac{1}{2} \times U + \langle \vec{D}, \vec{D}' \rangle \quad (2)$$

$$\vec{D}' = \begin{cases} \frac{\vec{S}'_c - \vec{S}_c}{\|\vec{S}'_c - \vec{S}_c\|} & \text{if } \vec{S}'_c \neq \vec{S}_c \\ \vec{0} & \text{otherwise} \end{cases} \quad (3)$$

After the update of U (after setting U to U'), depending on the situation, the step size is doubled or halved:

- if the procedure in Figure 4 cannot find a better candidate solution in the neighborhood of \vec{S}_c , *i.e.*, if $\vec{S}'_c = \vec{S}_c$, then R is halved;
- if the agent appears to be moving in a forward direction, *i.e.*, if $U > r_e$, then R is doubled to accelerate the convergence of the agent, and U is set to 0. This means that U cannot take values higher than r_e .

To prevent U from having high negative values, U is constrained to be higher or equal to $-r_e$.

The stopping criterion of the local search is presented in Figure 5, where δ_{ph} and δ_{pl} are two parameters of MLSDO. If the stopping criterion is satisfied, then the procedure returns *true*, otherwise, it returns *false*. As we can see, if the current solution of an agent is the best solution found by MLSDO since the last change in the objective function, then we use a higher precision δ_{ph} in the stagnation criterion of its local search, otherwise we use a lower precision δ_{pl} . We choose δ_{ph} to be not larger than δ_{pl} . In this way, we prevent the fine-tuning of low quality solutions, which could lead to a waste of fitness function evaluations; only the best solution found by the algorithm is fine-tuned.

B. The diversity maintaining strategy

If an agent has found a local optimum, then this optimum is transmitted to memory through the coordinator. Afterwards, the coordinator gives the agent its new position (see subsection III-C), in order to perform a new local search (and R is initialized to r_e).

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 is attributed to each agent. This exclusion radius is the parameter r_e . Hence, if an agent detects one or several other agents at a distance lower than r_e , then only the agent with the best fitness, among the detected agents including the agent having detected them, is allowed to continue its search. All the other agents have to be relocated.

C. The relocation of the agents

If an agent has terminated its local search (it has found an optimum), or if it has been found too close to other agents (see Figure 2), then the coordinator can either destroy the agent, or let the agent start a new local search at a given position. It is performed according to the procedure in Figure 6.

stoppingCriterion

Inputs: $\vec{S}'_c, \vec{S}_w, \vec{S}_c, R, r_l, \delta_{ph}, \delta_{pl}$

local variables: p

```

if  $R < r_l$  then
  if  $\vec{S}'_c \neq \vec{S}_c$  then
     $p \leftarrow |\text{fitness}(\vec{S}'_c) - \text{fitness}(\vec{S}_c)|$ 
  else
     $p \leftarrow |\text{fitness}(\vec{S}_w) - \text{fitness}(\vec{S}_c)|$ 
  end
  if  $\vec{S}_c$  is the best solution found since the last change in the
  objective function then
    if  $p \leq \delta_{ph}$  then
      return true
    end
  else
    if  $p \leq \delta_{pl}$  then
      return true
    end
  end
end
return false

```

Fig. 5. Procedure that tests the stopping criterion of an agent. *fitness* is a function that returns the value of the objective function of a given solution. \vec{S}'_c is the candidate solution found to replace the current solution \vec{S}_c of the agent. \vec{S}_w is the worst tested candidate solution. R is the step size of the agent. r_l is the initial step size of tracking agents. δ_{ph} and δ_{pl} are the highest and the lowest precision parameters of the stagnation criterion, respectively.

relocateOrDestroyAgent

Inputs: N, n_a, d, A_m, A_i, r_e

local variables: d_0, \vec{S}_{new}

```

if  $N > n_a$  then
  destroy the agent
end
 $\{d_0, \vec{S}_{new}\} \leftarrow \text{newInitialSolution}(d, A_m, A_i)$ 
if  $(N > 1)$  and  $(d_0 \leq r_e)$  then
  destroy the agent
else
  relocate the agent at  $\vec{S}_{new}$  with an initial step size of  $r_e$ 
end

```

Fig. 6. Procedure that destroys or relocates an agent. N is the number of currently existing agents. n_a is the maximum number of exploring agents. d is the dimension of the search space. A_m is the archive of the local optima found by the agents. A_i is the archive of the last initial positions of the agents. r_e is the initial step size of exploring agents.

We can note that this procedure makes use of two archives: A_m and A_i . The archive A_m contains the saved optima (its capacity is equal to n_m , see section III-E). The archive A_i saves the last n_m initial positions of agents to be created or relocated. Each time a change in the objective function is detected, the archive A_i is cleared. This procedure produces a

newInitialSolution

Inputs: d, A_m, A_i
local variables: $\vec{S}, d_1, d_2, a_i, \vec{S}_c, \vec{S}_i$
 $d_0 \leftarrow -\infty$
repeat $10 \times d$ times
 $\vec{S} \leftarrow$ a random solution uniformly chosen in the search space
 $d_1 \leftarrow +\infty$
 for each agent a_i **do**
 $\vec{S}_c \leftarrow$ the current solution of a_i
 $d_2 \leftarrow$ the distance between \vec{S} and \vec{S}_c
 if $d_2 < d_1$ **then**
 $d_1 \leftarrow d_2$
 end
 end
 for each $\vec{S}_i \in (A_m \cup A_i)$ **do**
 $d_2 \leftarrow$ the distance between \vec{S} and \vec{S}_i
 if $d_2 < d_1$ **then**
 $d_1 \leftarrow d_2$
 end
 end
 if $d_1 > d_0$ **then**
 $d_0 \leftarrow d_1$
 $\vec{S}_{new} \leftarrow \vec{S}$
 end
end
return $\{d_0, \vec{S}_{new}\}$

Fig. 7. Procedure to generate an initial solution \vec{S}_{new} for an agent. It also returns the distance d_0 between this solution, and the closest one in the set of current solutions of the agents, and of solutions stored in A_m (the archive of the found local optima) and A_i (the archive of the last initial positions of the agents). d is the dimension of the search space.

new position for the agent which has to be relocated, which is far from all the other agents, and from the solutions stored in A_m and A_i , by using the *newInitialSolution* procedure (see Figure 7). This heuristic generates several random locations uniformly distributed in the search space, selects one of them and returns it. The selection mechanism of this heuristic is as follows. For each generated location, the distance to the closest location in the set of current solutions of the agents, and of solutions stored in A_m and A_i , is calculated. Then, the generated location that has the greatest calculated distance is selected. The number of generated locations has been empirically set to $10 \times d$, where d is the dimension of the search space. It is a compromise between the computational cost, and the accuracy of the heuristic. If the new position \vec{S}_{new} for the agent is in an unexplored zone of the search space (if it is not too close to another agent, to an archived optimum or to an archived initial position), then the agent is relocated at \vec{S}_{new} . Otherwise, the search space is considered saturated and the coordinator destroys the agent. The decision of destroying the agent is also taken if more than n_a agents exist. It happens if agents are created to track archived optima

addAgents

Inputs: $n_a, n_c, N, m, A_m, r_l, r_e$
local variables: n_t, n_n, \vec{O}_{best}
 $n_t \leftarrow \max(0, \min(n_a + n_c - N, n_c, m))$
 $n_n \leftarrow \max(0, \min(n_a - N - n_t, m - n_t))$
repeat n_t times
 $\vec{O}_{best} \leftarrow$ the best optimum in A_m
 $A_m \leftarrow A_m - \{\vec{O}_{best}\}$
 create an agent with initial solution \vec{O}_{best} and initial step size r_l
end
repeat n_n times
 $\vec{O}_{best} \leftarrow$ the best optimum in A_m
 $A_m \leftarrow A_m - \{\vec{O}_{best}\}$
 create an agent with initial solution \vec{O}_{best} and initial step size r_e
end

Fig. 8. Procedure to create additional agents after a change, to track the best archived optima and to make the number N of existing agents at least equal to n_a (the maximum number of exploring agents). *max* and *min* are functions that return the maximum and the minimum value among several given values, respectively. n_c is the maximum number of tracking agents. m is the number of local optima currently stored in the archive A_m . r_l and r_e are the initial step sizes of tracking and exploring agents, respectively.

after the detection of a change in the objective function (see subsection III-D).

D. The change detection and the tracking of the optima

The coordinator detects the changes in the objective function. This detection is performed when all the agents have completed one step of their search procedure, *i.e.*, when all the agents are in a SYNCHRONIZATION state (see subsection II-D). Changes in the objective function are detected by reevaluating the fitness of a randomly chosen agent or archived optimum, and comparing it to its previous value. If these values are different, a change is supposed to have occurred, and the following actions are taken: the fitnesses of all agents and archived optima are reevaluated; then, the procedure of the creation of additional agents (Figure 8) is executed.

These additional agents are initialized using the best optima in A_m as initial solutions. Each time an agent is created, the optimum used to initialize it is removed from A_m . The maximum number of tracking agents to create (to track optima when a change is detected) is n_c . After creating the tracking agents, if the number N of currently existing agents is lower than n_a , then at most $n_a - N$ exploring agents are created.

E. Archive management

The memory manager maintains the archive A_m of local optima found by the agents. This archive must be bounded, its size is fixed to a number n_m of entries. We propose the expression (4) to calculate the value of n_m , where the *round* function rounds a number to the nearest integer, r_e is the

```

replaceDominatedOptima


---


Inputs:  $\vec{O}_c, A_m, r_l, r_e$ 
local variables:  $A_{sub}, \vec{O}_{best}, \vec{O}_i$ 
 $A_{sub} \leftarrow \emptyset$ 
 $\vec{O}_{best} \leftarrow \vec{O}_c$ 
for each  $\vec{O}_i \in A_m$  do
  if  $\|\vec{O}_c - \vec{O}_i\| \leq \sqrt{r_l \times r_e}$  then
     $A_{sub} \leftarrow A_{sub} \cup \{\vec{O}_i\}$ 
    if  $\vec{O}_i$  is strictly better than  $\vec{O}_{best}$  then
       $\vec{O}_{best} \leftarrow \vec{O}_i$ 
    end
  end
end
 $A_m \leftarrow A_m - A_{sub}$ 
 $A_m \leftarrow A_m \cup \{\vec{O}_{best}\}$ 
return  $A_m$ 

```

Fig. 9. Procedure that replaces the dominated optima. \vec{O}_c is the newly found optimum. A_m is the archive of the local optima found by the agents. r_l and r_e are the initial step sizes of tracking and exploring agents, respectively.

exclusion radius of the agents and d is the dimension of the search space. This expression was defined empirically.

$$n_m = \text{round}\left(\frac{d}{r_e}\right) \quad (4)$$

We introduce a flag *isNotUpToDate* that indicates if a change in the objective function occurred since the detection of a given stored optimum: if a change occurred, it returns *true*; otherwise, it returns *false*.

If the archive is full, then we use the following conditions to update the archive:

- 1) If the new optimum, denoted by \vec{O}_c , is better than the worst optimum in the archive, or its value is at least equal to the one of this worst optimum, then:
 - a) If there is one or several optima in the archive where *isNotUpToDate* returns *true*, then the worst of them is replaced by \vec{O}_c ;
 - b) otherwise, the worst optimum of the archive is replaced by \vec{O}_c .
- 2) If there is one or several optima in the archive that are “too close” to \vec{O}_c (an archived optimum is considered “too close” to \vec{O}_c if it lies at a distance from \vec{O}_c lower or equal to the geometric average of r_l and r_e), then all these optima close to each other are considered to be dominated by the best of them. Thus, this subset of solutions is replaced by only their best one. The different steps of this replacement are in Figure 9.

IV. BENCHMARK SET

The performances of MLSDO are evaluated using the benchmark generator for the IEEE WCCI-2012 competition

Parameter	Value
Dimension (fixed)	5
Dimension (changed)	[2, 15]
Change frequency	50000
Number of changes	60
Number of peaks (fixed)	10
Number of peaks (changed)	[10, 50]

TABLE I
MAIN PARAMETERS USED FOR THE COMPETITION.

on evolutionary computation for dynamic optimization problems. It uses the Generalized Dynamic Benchmark Generator (GDBG) [4], and it is based on the benchmark used during the IEEE CEC’09 competition on dynamic optimization [5]. Six functions are used to create this benchmark:

- F₁: rotation peak function
- F₂: composition of Sphere’s function
- F₃: composition of Rastrigin’s function
- F₄: composition of Griewank’s function
- F₅: composition of Ackley’s function
- F₆: hybrid composition function

A total of eight dynamic scenarios with different degrees of difficulty are proposed:

- T₁: small step change (a small displacement)
- T₂: large step change (a large displacement)
- T₃: random change (Gaussian displacement)
- T₄: chaotic change (logistic function)
- T₅: recurrent change (a periodic displacement)
- T₆: recurrent with noise (the same as above, but the optimum never returns exactly to the same point)
- T₇: changing the dimension of the problem
- T₈: changing the number of peaks

The basic parameters of the benchmark are given in Table I.

There are 60 test cases that correspond to the combinations of the six problems with the eight change scenarios using a changing ratio equal to 1.0, and to the combinations of the problem F₁ with the first six change scenarios using a changing ratio equal to 0.3 and 0.7. The changing ratio is a value in [0, 1], and it gives the number of peaks that are allowed to change. For each test case, the mean value of the absolute error and the corresponding standard deviation are recorded, denoted by *mean* and *STD*, respectively.

Then, a mark is calculated for each test case, denoted by *performance_k*. The sum of all marks *performance_k* gives a score, denoted by *performance*, that corresponds to the overall performance of the tested algorithm. The maximum value of this score is 60 for the competition.

The values of *mean*, *STD* and *performance_k* are calculated according to the method proposed in [6].

V. RESULTS AND DISCUSSION

In [2], MLSDO has been evaluated using the benchmark functions provided for the IEEE CEC’09 competition on

Name	Type	Interval	Value	Short description
r_e	real	$(0, 1]$	0.1	exclusion radius of the agents, and initial step size of exploring agents
r_l	real	$(0, r_e)$	0.07	initial step size of tracking agents
δ_{ph}	real	$[0, \delta_{pl}]$	0.001	highest precision parameter of the stagnation criterion of the agents local searches
δ_{pl}	real	$[\delta_{ph}, +\infty]$	1.0	lowest precision parameter of the stagnation criterion of the agents local searches
n_a	integer	$[1, 10]$	3	maximum number of exploring agents
n_c	integer	$[0, 20]$	1	maximum number of tracking agents created after the detection of a change

TABLE II
MLSDO PARAMETER SETTING FOR THE SET OF BENCHMARK FUNCTIONS AT HAND.

dynamic optimization. The overall performance of MLSDO for the CEC'09 competition is equal to 81.28 (a score out of 100), which is better than the score obtained by the winner of the CEC'09 competition, equal to 69.73 [7].

In the following subsections, the parameter setting and the results obtained by MLSDO for the IEEE WCCI-2012 competition are presented and discussed.

A. Parameter setting of MLSDO

Table II summarizes the six parameters of MLSDO that the user has to define. The given values are suitable for the set of test functions provided for the competition. These values were fixed empirically, and used to perform all our experiments. Each parameter has been fitted manually, one after the other, since there is no important correlation between them. A total of 26 possible sets of values for the parameters has been tested, from which we selected the one that maximizes the overall performance. The first tested set of values is the one used for the benchmark functions of the IEEE CEC'09 competition on dynamic optimization ($r_e = 0.1$, $r_l = 0.005$, $\delta_{ph} = 0.001$, $\delta_{pl} = 1.5$, $n_a = 5$, $n_c = 0$). Using this set of values, we get an overall performance equal to 56.192 for the IEEE WCCI-2012 competition. Then, by testing lower and greater values for each parameter, we reach a better overall performance, equal to 56.424.

To fit the parameters of MLSDO, one can take into account the following considerations. The number of agents n_a cannot be too high, because the convergence needs to be fast. Having too many agents exploring the search space slows down indeed the individual convergence of each agent. The lowest precision parameter δ_{pl} needs to be correctly adapted to the objective function and to its change severity. A low value for δ_{pl} prevents the agents from widely exploring the search space for a fast

System	Linux (Red Hat)
CPU	2 × Intel® Xeon® X5650 (6 cores, 2.66 GHz)
RAM	48 GB
Language	C++
Algorithm	Multiple Local Search algorithm for Dynamic Optimization (MLSDO)

TABLE III
COMPUTER CONFIGURATION.

changing objective function, since they will spend too many evaluations on fine-tuning their current solution, and too few ones on exploring other zones of the search space. Hence, the lower the number of evaluations between two changes is, the higher the value of δ_{pl} should be. This means that the compromise between a high precision of the found optima (intensification) and a wide exploration of the search space (diversification) needs to be in favor of diversification for fast changing objective functions. The exclusion radius r_e should match the radius of the attraction zone of an optimum and r_l has to be lower than r_e . The parameter n_c corresponds to the number of archived optima that need to be tracked at every change of the objective function. If the objective function changes strongly enough, and the positions of the optima can move to any random location in the search space, then n_c tends to 0. On the contrary, if the changes are smooth enough, the tracking of the optima is possible, then, the value of n_c corresponds to the number of promising optima to track.

B. Experimental results

The configuration of the computer used for simulation is given in Table III, with other technical details. MLSDO was tested using the test platform provided for the competition, based on the *EALib* library.

Table IV and Table V present the results of MLSDO for the mean value (*mean*) and the corresponding standard deviation (*STD*) of each test case. Table VI shows the performance of MLSDO on each test case, and the overall performance.

In Table VI, by comparing the sums of scores obtained by MLSDO for each problem, using a changing ratio equal to 1.0, we can see that MLSDO performs best for the problems F_1 and F_5 . As the diversification process is well controlled in our algorithm, and the cited objective functions are smooth, MLSDO finds quickly the global optimum. The worst results are obtained for the problem F_3 , which is based on the Rastrigin function. This is an highly multimodal function, where each local optimum has a large attraction zone. Hence, it can be harder for MLSDO to find the global optimum for this problem.

We can see in Table VII that MLSDO performs worst on the change scenarios T_7 and T_8 . These change scenarios are the only ones that make the dimension of the search space, or the number of peaks, vary. They can be indeed hard kinds of changes. The best results are obtained for the change scenario T_1 . This scenario involves small displacements of the global optimum in the search space, that can therefore be easier to track. The varying performance of MLSDO across the different

Changing ratio	T ₁	T ₂	T ₃	T ₄	T ₅	T ₆	T ₇	T ₈
	<i>mean</i> ± <i>STD</i>	<i>mean</i> ± <i>STD</i>	<i>mean</i> ± <i>STD</i>	<i>mean</i> ± <i>STD</i>	<i>mean</i> ± <i>STD</i>	<i>mean</i> ± <i>STD</i>	<i>mean</i> ± <i>STD</i>	<i>mean</i> ± <i>STD</i>
0.3	0.117 ± 0.213	0.677 ± 0.426	0.180 ± 0.075	0.062 ± 0.121	0.012 ± 0.020	0.154 ± 0.320	—	—
0.7	0.571 ± 0.395	1.145 ± 0.692	0.846 ± 1.098	0.045 ± 0.033	0.011 ± 0.018	0.086 ± 0.049	—	—
1.0	0.583 ± 1.315	1.129 ± 1.110	1.736 ± 0.623	0.003 ± 0.006	0.046 ± 0.025	0.013 ± 0.007	1.909 ± 3.838	0.859 ± 2.449

TABLE IV
MEAN VALUES AND *STD* FOR THE PROBLEM F₁ WITH CHANGING RATIOS 0.3, 0.7 AND 1.0, RESPECTIVELY.

Problem	T ₁	T ₂	T ₃	T ₄	T ₅	T ₆	T ₇	T ₈
	<i>mean</i> ± <i>STD</i>	<i>mean</i> ± <i>STD</i>	<i>mean</i> ± <i>STD</i>	<i>mean</i> ± <i>STD</i>	<i>mean</i> ± <i>STD</i>	<i>mean</i> ± <i>STD</i>	<i>mean</i> ± <i>STD</i>	<i>mean</i> ± <i>STD</i>
F ₂	0.225 ± 0.268	0.079 ± 0.153	0.628 ± 0.445	0.002 ± 0.003	0.913 ± 0.694	0.022 ± 0.060	0.882 ± 2.817	4.142 ± 6.094
F ₃	0.173 ± 0.499	4.143 ± 5.863	4.648 ± 6.037	1.787 ± 2.951	0.508 ± 0.849	5.338 ± 9.577	146.003 ± 274.080	2.993 ± 4.479
F ₄	0.240 ± 0.399	0.334 ± 0.520	1.109 ± 1.690	0.020 ± 0.079	2.372 ± 4.582	0.160 ± 0.338	1.176 ± 3.252	4.620 ± 6.624
F ₅	0.010 ± 0.017	0.014 ± 0.015	0.005 ± 0.009	0.002 ± 0.003	0.005 ± 0.015	0.010 ± 0.013	0.086 ± 0.359	0.073 ± 0.257
F ₆	0.179 ± 0.437	1.375 ± 1.308	0.737 ± 1.008	0.113 ± 0.350	0.201 ± 0.241	0.689 ± 0.713	1.967 ± 4.299	3.941 ± 5.831

TABLE V
MEAN VALUES AND *STD* FOR THE PROBLEMS F₂ TO F₆.

Problem	Changing ratio	T ₁	T ₂	T ₃	T ₄	T ₅	T ₆	T ₇	T ₈	Sum
F ₁	0.3	0.998	0.996	0.999	0.999	0.999	0.998	—	—	5.989
	0.7	1.000	1.000	0.978	0.998	0.998	0.998	—	—	5.973
	1.0	0.985	0.975	0.982	0.998	0.996	0.995	0.955	0.978	7.864
F ₂	1.0	0.976	0.976	0.962	0.987	0.908	0.977	0.930	0.840	7.557
F ₃	1.0	0.946	0.788	0.806	0.854	0.891	0.815	0.631	0.827	6.558
F ₄	1.0	0.952	0.944	0.925	0.962	0.850	0.943	0.907	0.815	7.298
F ₅	1.0	0.976	0.975	0.976	0.974	0.976	0.974	0.952	0.962	7.766
F ₆	1.0	0.968	0.932	0.941	0.956	0.962	0.925	0.900	0.836	7.419
The overall performance										56.424

TABLE VI
OVERALL PERFORMANCE.

	T ₁	T ₂	T ₃	T ₄	T ₅	T ₆	T ₇	T ₈
Sum	5.802	5.591	5.592	5.731	5.583	5.628	5.276	5.259

TABLE VII
SUM, FOR EACH CHANGE SCENARIO, OF THE SCORES OBTAINED FOR THE PROBLEMS F₁ TO F₆, USING A CHANGING RATIO EQUAL TO 1.0.

change scenarios shows that the algorithm is sensitive to the nature and the severity of the changes.

VI. CONCLUSION

The MLSDO algorithm, developed in order to solve a wide range of DOPs, has been presented. It is based on several coordinated local searches and on the archiving of the found local optima, in order to track them after a change in the objective function.

In works in progress, in order to reduce the number of parameters and increase the efficiency of MLSDO, we are analyzing the sensitivity of its parameters. One future plan is to make the critical parameters of MLSDO adaptive, *i.e.*, such that they will be automatically adjusted. Besides, as many real-world DOPs are multiobjective, MLSDO may also be adapted to the dynamic multiobjective optimization.

Our goal in this paper was to perform the experiments and to present the obtained results as required by the technical report for the IEEE WCCI-2012 competition [6]. The overall performance (see Table VI) obtained by MLSDO is 56.424.

REFERENCES

[1] J. Lepagnot, A. Nakib, H. Oulhadj, and P. Siarry, "Brain cine-MRI registration using MLSDO dynamic optimization algorithm," in *Proc. 9th*

Metaheuristics International Conference (MIC 2011), vol. 1, Udine, Italy, 2011, pp. 241–249.

[2] J. Lepagnot, A. Nakib, H. Oulhadj, and P. Siarry, "Brain cine-MRI Sequences Free-Form Deformations and MLSDO Dynamic Optimization Algorithm," to appear in *Proc. International Conference on Learning and Intelligent Optimization (LION6)*, Paris, France, January 2012.

[3] J. Lepagnot, A. Nakib, H. Oulhadj, and P. Siarry, "A new multiagent algorithm for dynamic continuous optimization," *International Journal of Applied Metaheuristic Computing*, vol. 1, no. 1, pp. 16–38, 2010.

[4] C. Li and S. Yang, "A generalized approach to construct benchmark problems for dynamic optimization," in *Proc. 7th International Conference on Simulated Evolution and Learning*. Melbourne, Australia: Springer, 2008, pp. 391–400.

[5] C. Li, S. Yang, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H.-G. Beyer, and P. N. Suganthan, "Benchmark generator for CEC 2009 competition on dynamic optimization," University of Leicester, University of Birmingham, Nanyang Technological University, Tech. Rep., 2008.

[6] C. Li, S. Yang, and D. A. Pelta, "Benchmark generator for the IEEE WCCI-2012 competition on evolutionary computation for dynamic optimization problems," China University of Geosciences, Brunel University, University of Granada, Tech. Rep., 2011.

[7] J. Brest, A. Zamuda, B. Boskovic, M. S. Maucec, and V. Zumer, "Dynamic optimization using self-adaptive differential evolution," in *Proceedings of the IEEE Congress on Evolutionary Computation*. Trondheim, Norway: IEEE, 2009, pp. 415–422.