Journal of Heuristics manuscript No.
(will be inserted by the editor)

A Multiple Local Search Algorithm for Continuous Dynamic Optimization

Julien Lepagnot · **Amir Nakib** ·
Hamouche Oulhadj · **Patrick Siarry**

Received: date / Accepted: date

Abstract Many real-world optimization problems are dynamic (time dependent) and require an algorithm that is able to track continuously a changing optimum over time. In this paper, we propose a new algorithm for dynamic continuous optimization. The proposed algorithm is based on several coordinated local searches and on the archiving of the optima found by these local searches. This archive is used when the environment changes. The performance of the algorithm is analyzed on the Moving Peaks Benchmark and the Generalized Dynamic Benchmark Generator. Then, a comparison of its performance to the performance of competing dynamic optimization algorithms available in the literature is done. The obtained results show the efficiency of the proposed algorithm.

Keywords dynamic · non-stationary · time-varying · continuous optimization · multi-agent · metaheuristic · Moving Peaks

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. For instance, changes can be due to machine breakdowns, to the changing quality of raw materials, or to the appearance of new jobs to be added to a schedule. In dynamic environments, the goal is not only to locate the global optimum, but also to follow it as closely as possible.

A dynamic optimization problem can be expressed as in (1), where $f(\mathbf{x},t)$ is the objective function of a minimization problem, $h_j(\mathbf{x},t)$ denotes the j^{th} equality

Patrick Siarry
Laboratoire Images, Signaux et Systèmes Intelligents, LISSI, E.A. 3956
Université Paris-Est Créteil, 61 avenue du Général de Gaulle, 94010 Créteil, France
E-mail: siarry@u-pec.fr

constraint and $g_k(\mathbf{x},t)$ denotes the k^{th} inequality constraint. All of these functions may change over time (iterations), as indicated by the dependence on the time variable t.

$$\begin{aligned} &\min\ f(\mathbf{x},t) \\ &\text{s.t.}\ \ h_j(\mathbf{x},t) = 0 \text{ for } j = 1,2,...,u \\ &\qquad g_k(\mathbf{x},t) \leq 0 \text{ for } k = 1,2,...,v \end{aligned} \tag{1}$$

The performance of dynamic optimization algorithms in the literature is improving, and many research directions are left to be further investigated in order to obtain even more efficient algorithms. In this paper, a new algorithm for dynamic continuous optimization is proposed, called MLSDO (*Multiple Local Search algorithm for Dynamic Optimization*). It belongs to the class of cooperative search strategies for DOPs. 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. The set of heuristics used to coordinate and control all the local searches, and to manage the archived optima, constitutes a new efficient way to deal with DOPs.

The rest of this paper is organized as follows. Section 2 discusses some related work. Section 3 describes the fundamentals of the proposed MLSDO algorithm. Section 4 explains in detail each strategy used in MLSDO. Section 5 presents the benchmark sets used to test the algorithm. Experimental results are discussed in Section 6. Conclusions and work under progress are presented in section 7. A nomenclature of all the variables used in this paper is given in Appendix.

2 Related work

Almost all algorithms for DOPs are population-based metaheuristics, which are generally bioinspired algorithms. Most bioinspired algorithms belong to the classes of evolutionary algorithms (EAs) and particle swarm optimization (PSO), though ant colony optimization and artificial immune systems have also been investigated [8, 10, 15, 34]. In order to perform better on DOPs, some authors have tried hybrid algorithms, like Lung and Dumitrescu in [24] and [25], who propose hybrid PSO/EAs algorithms. However, the results of those algorithms are not significantly different from those of pure PSO or EAs algorithms. The use of local search as a main feature of a dynamic optimization algorithm has also been studied. We will now describe competing dynamic metaheuristics that have been proposed in the literature for continuous environments, and focus on the cooperative techniques used in each of these algorithms. Then, we highlight the main strategies and cooperative techniques used to deal with DOPs.

A significant number of existing dynamic optimization algorithms are based on EA principles. EAs can be indeed well suited to optimization in changing environments, since they are inspired by the principles of natural evolution. It has been shown

that multipopulation EAs allow to provide good results on multimodal DOPs. Promising results have been obtained by a self-organizing scouts algorithm proposed by Branke *et al.* in [6]. In this algorithm, a parent population searches over the entire search space while child populations track the local optima. The creation and merging of these child populations, as well as the adjustment of the number of individuals in each population, are typical examples of cooperative techniques that can be used in a multipopulation algorithm for DOPs. In [26], Mendes and Mohais propose a multipopulation differential evolution (DE[1]) algorithm based on some techniques to maintain diversity. Especially, if the best individuals of several subpopulations have a separating distance less than a predefined threshold, called the *exclusion radius*, then only the population with the highest performance is kept, and the others are reinitialized. The use of an exclusion radius is another common technique to coordinate several subpopulations. Among EAs, this algorithm achieves the best results on the Moving Peaks Benchmark (MPB) [4, 5]. Another multipopulation DE based algorithm is proposed by Brest *et al.* in [7] that also makes use of an exclusion radius to reinitialize overlapping subpopulations. It obtains the best results among EAs on the Generalized Dynamic Benchmark Generator (GDBG) [18, 20].

The different techniques used in EAs to deal with DOPs are classified in [13] into the following groups:

1. **Generating diversity after a change:** if a change in the environment is detected, then explicit actions are taken to increase diversity and to facilitate the shift to the new optimum.
2. **Maintaining diversity throughout the run:** convergence is avoided over the time and it is hoped that a spread-out population can adapt to changes more efficiently.
3. **Memory-based approaches:** the optimization algorithm is supplied with a memory to be able to further recall useful information from the past. In practice, good solutions are archived in order to be reused when a change is detected. More sophisticated memory-based techniques also exist [37].
4. **Multipopulation approaches:** dividing up the population into several subpopulations, distributed on different optima, allows tracking multiple optima simultaneously and increases the probability of finding new ones.

Another widely used class of algorithms for DOPs is PSO. PSO is a population-based approach in which simple software agents, called particles, move in the search space. The particle dynamics are inspired by models of swarming and flocking [14]. Dynamic PSO based algorithms are mainly multi-swarm, *i.e.* they make use of several sub-swarms, as several sub-populations can be used in EAs. A multi-swarm algorithm using a speciation process is proposed by Parrott and Li in [21, 30], where spatially close particles form a particular species. Each species corresponds to a sub-swarm and, if the number of particles in a sub-swarm is higher than a predefined threshold, then the worst particles in this sub-swarm are reinitialized. This technique can be

[1] The strategy of DE consists in generating a new position for an individual, according to the differences calculated between other randomly selected individuals.

also considered as a cooperative one that enables the regulation of the number of particles in a sub-swarm. Several improved variants of this algorithm are proposed in [1, 22, 31]. Du and Li propose another PSO based algorithm in [9]. Here, two swarms of particles are used: the first one is for diversification, and the second is for intensification. The exchanges of information about the velocities and positions of the particles between the swarms enable their cooperation. Li and Yang propose in [19] a multi-swarm approach that makes use of clustering techniques to create sub-swarms, and overlapping sub-swarms are merged. Best solutions found until a change occurs are archived, in order to be added as new particles if a change is detected. It is yet the only PSO algorithm analyzed using GDBG. A simplified version of this algorithm is benchmarked using MPB [36].

The use of concepts from the physics domain in PSO has been also widely investigated. In [2], Blackwell and Branke 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) do not follow a classical trajectory, but are instead randomized within a ball centered on the swarm attractor. If several swarms converge to the same local optima, then an exclusion radius is used in order to randomize the worst ones. Both of these algorithms place their swarms on each of the localized optima, then each swarm tracks a different optimum. The same approach is used in [3], [22] and [29], hybridized with other techniques to increase the diversity and to track the optima. Among PSO algorithms, the best results obtained on MPB are achieved by Novoa *et al.* [29], thanks to the use of a simple controlling mechanism that determines how the position and velocity of a particle have to be updated: if a particle performs a fixed number of successive non improving moves in the search space, then it is relocated at the best position found by its swarm, and its velocity is multiplied by a random number in $[0, 0.5]$. Otherwise, its position and velocity are updated using the classical equations of PSO. This way, the particles that tend to waste evaluations in unpromising regions of the search space are relocated in more promising ones.

Some authors investigated also the use of local search as a main feature of a dynamic optimization algorithm. We can cite [39], where Zeng *et al.* propose an algorithm based on the use of local searches to explore the search space. The local optima found are archived, in order to be tracked when a change occurs, using additional local searches. A specialized local search procedure is proposed in order to provide a fast convergence to the local optima. In [28], Moser and Hendtlass use extremal optimization (EO) to determine the initial solution of a local search procedure. 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 initial solution of a hill-climbing phase, in order to fine-tune the solution. Finally, the solution is stored in memory, and the algorithm is applied again on another randomly generated solution. An improved and generalized variant of this algorithm is proposed in [27], termed as Hybridised EO. Compared to the algorithm of Zeng *et*

al. [39], where the exploring local searches are randomly initialized, Hybridised EO makes use of a special seeding strategy to initialize the local searches in promising areas of the search space. However, this algorithm has been specifically designed for MPB and no study has been done yet to determine whether there is a potential of wider applicability. In [16, 17], Lepagnot *et al.* propose a multi-agent centralized cooperative algorithm for DOPs, called MADO. Local searches are used also to explore the search space, and to track the local optima stored in memory. As Hybridised EO, it makes use of a special seeding strategy to initialize the local searches in promising areas of the search space. However, in MADO, the local searches are performed by a population of agents that are coordinated. In [32, 33], Pelta *et al.* propose a multi-agent decentralized cooperative strategy, where multiple agents cooperate to improve a set of solutions stored on a grid. Then, Gonzalez *et al.* presented a new centralized cooperative strategy for DOPs that uses several tabu search based local searches [12]. As in MADO, these local searches are controlled by a coordinator that keeps a memory of the found local optima. Besides, Wang *et al.* propose in [35] a memetic algorithm based on the cooperation and competition of two local search procedures, combined with diversity maintaining techniques.

In this paper, the proposed MLSDO algorithm makes also use of the main existing cooperative techniques in a centralized framework, implemented in a different way, *i.e.* a dynamic number of agents perform local searches in parallel; the found local optima are archived in order to be tracked when a change is detected; a seeding strategy is used to initialize the exploring local search agents in promising, or unexplored, areas of the search space; an exclusion radius is used to prevent several local searches to converge to the same local optimum. These techniques provide a maintained and enhanced diversity throughout the execution of the algorithm, in order to widely explore the search space and detect the local optima. They also enable a fast convergence to the local optima, using several parallel local searches, and their tracking. This way, the algorithm can adapt and react to changes more efficiently. In [17], we proposed to combine these main ideas to solve DOPs in a preliminary work. It is obvious that these techniques can be implemented in different ways that can lead to different performances. In this paper, new heuristics are proposed to implement them, and it constitutes a new efficient way to deal with DOPs. In MLSDO, our motivation is to propose more suitable heuristics for dynamic optimization. Each heuristic in MLSDO is designed by a trial and error approach, using the main two benchmarks available in dynamic optimization to evaluate, and retain an algorithm. It leads to novel algorithms as the seeding mechanism used to initialize local search agents, the stopping criterion of their local search procedure, as well as an adaptation mechanism used to adapt the step size of the performed local searches. All these algorithms will be detailed in Section 4, and analyzed in Section 6, along with a comparison of MLSDO to other competing algorithms based on cooperative techniques.

3 Proposed 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.

3.1 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 $i \in \{1,...,d\}$. Then, the unit vectors $(\mathbf{e}_i)$ in the direction of each axis of the Cartesian coordinates system are scaled, in order to produce modified basis vectors $(\mathbf{u}_i)$, defined as $\{\mathbf{u}_1 = \Delta_1 \mathbf{e}_1, \mathbf{u}_2 = \Delta_2 \mathbf{e}_2, ..., \mathbf{u}_d = \Delta_d \mathbf{e}_d\}$. This change in basis transforms a hyper-rectangular search space into a hyper-square search space, according to (2), 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.

$$x'_i = \frac{x_i}{\Delta_i} \text{ for } i = 1,2,...,d \tag{2}$$

3.2 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 environment, a dynamic optimization algorithm needs to keep track of many of the found local optima, since one of them can become the new global optimum after a change occurs in the objective function. Thus, we propose to 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.

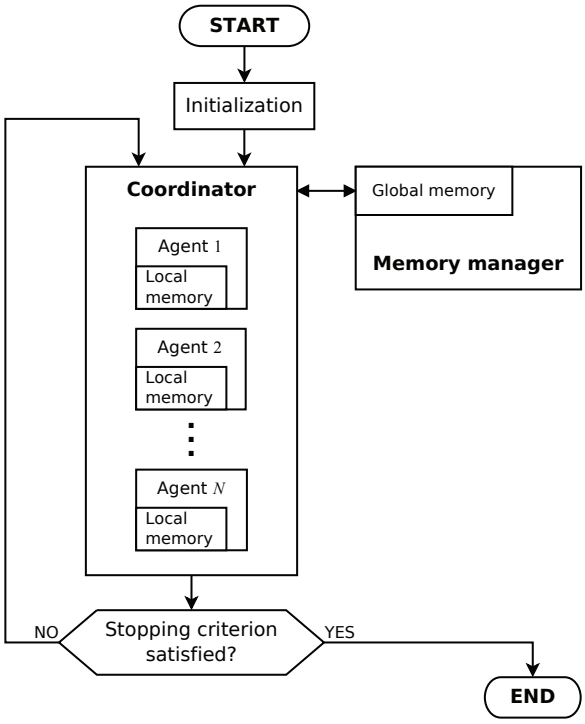

Fig. 1 Overall scheme of MLSDO. N is the number of currently existing agents. Each agent performs a search procedure described in subsection 3.4.

The initialization step in Figure 1 mainly consists in computing the initial set of agents, as described in the next subsection. The stopping criterion depends on the optimization problem.

3.3 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. The implementation details of this heuristic are given in subsection 4.3.

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.

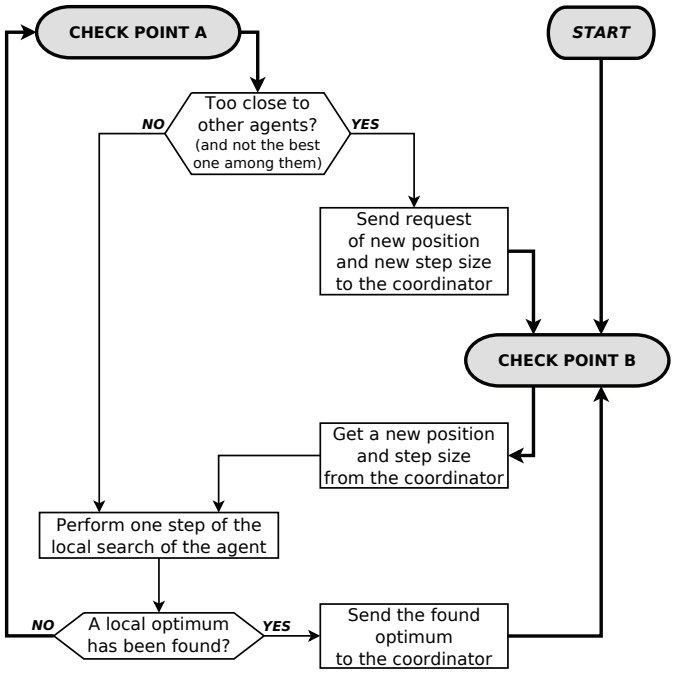

Fig. 2 Flowchart of the main procedure of a MLSDO agent.

3.4 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 "CHECK POINT A" and "CHECK POINT 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 CHECK POINT A (CHECK POINT B, respectively), then it resumes its execution on CHECK POINT A (CHECK POINT B, respectively). This special state allows the parallel execution of the agents.

4 The optimization strategies used in MLSDO

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

4.1 The exploration strategy of the agents

MLSDO agents explore the search space step-by-step, moving from their current solution $\mathbf{S_c}$ to a better one $\mathbf{S'_c}$ in their neighborhood, until they reach a local optimum.

initializeAgent

Inputs: $\mathbf{S_{new}}, r_{new}$
local variables: $\varnothing$
$\mathbf{S_c} \leftarrow \mathbf{S_{new}}$
Evaluate $\mathbf{S_c}$
$\mathbf{D} \leftarrow \mathbf{0}$
$R \leftarrow r_{new}$
$U \leftarrow 0.0$
return $\{\mathbf{S_c}, \mathbf{D}, R, U\}$

Fig. 3 Procedure that initializes the local search of an agent. $\mathbf{S_{new}}$ and r_{new} are the initial solution and the initial step size of the agent, respectively. $\mathbf{S_c}$ is the current solution of the agent. $\mathbf{D}$ is the direction vector of the last displacement of the agent. R is the step size of the agent. U is the value of the *cumulative dot product* (see equation 3). The function *Evaluate* computes the value of the objective function of a given solution and assigns this value to the solution. $\mathbf{0}$ is a vector of d zeros.

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.

An agent has a step size R, adapted during its local search, and initialized to r_{new} (a real parameter in the interval $(0,1]$). However, an agent created to explore the search space requires a greater initial step size than a "tracking" agent. Thus, when initializing the local search of a new agent, the value of r_{new} depends on the condition of the creation of the agent. The parameter r_{new} is equal to r_l for a "tracking" agent, and to r_e otherwise, where r_l and r_e are two parameters to be fixed. If the local search of an agent has to be reinitialized, then r_{new} must be equal to r_e. The initialization of the local search of an agent is described in Figure 3. This procedure is called at the creation, and at the relocation of an agent.

At the initialization of the local search of an agent, in addition to the step size R, the current solution $\mathbf{S_c}$ of the agent is set to $\mathbf{S_{new}}$, where $\mathbf{S_{new}}$ is the new starting solution given by the coordinator. The direction vector $\mathbf{D}$ and the *cumulative dot product* U, used to adapt R, are set to the null vector and zero, respectively.

We focus now on the local search of a single agent, summarized in Figure 4 and described in detail below. The procedure in Figure 4 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*, *stoppingCriterion* and *updateStepSize* that are defined below.

At each step of its local search, the agent moves from its current solution $\mathbf{S_c}$ to the new candidate solution $\mathbf{S'_c}$ according to the mechanism in Figure 5. As we can see, two candidate solutions are evaluated per dimension of the search space, denoted by $\mathbf{S_{prev}}$ and $\mathbf{S_{next}}$. They stand in opposite directions from $\mathbf{S'_c}$ along an axis of the search

localSearch

Inputs: $\mathbf{S_c}, d, R, U, \mathbf{D}, r_l, \delta_{ph}, \delta_{pl}$
local variables: $stop$
$\{\mathbf{S'_c}, \mathbf{S_w}, \mathbf{D'}\} \leftarrow$ selectCandidate$(\mathbf{S_c}, d, R)$
$stop \leftarrow$ stoppingCriterion$(\mathbf{S'_c}, \mathbf{S_w}, \mathbf{S_c}, R, r_l, \delta_{ph}, \delta_{pl})$
$\{U, R\} \leftarrow$ updateStepSize$(\mathbf{S'_c}, \mathbf{S_c}, U, \mathbf{D}, \mathbf{D'}, R)$
if $\mathbf{S'_c} \neq \mathbf{S_c}$ **then**
  $\mathbf{S_c} \leftarrow \mathbf{S'_c}$
  $\mathbf{D} \leftarrow \mathbf{D'}$
end
if $stop = true$ **then**
  Stopping criterion (of the local search) satisfied: $\mathbf{S_c}$ is the local optimum found
end
return $\{\mathbf{S_c}, R, U, \mathbf{D}\}$

Fig. 4 Procedure that performs one step of the local search of an agent. $\mathbf{S_c}$ 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 3). $\mathbf{D}$ is the direction vector of the last displacement of the agent. r_l, δ_{ph} and δ_{pl} are parameters of the algorithm. The procedures *selectCandidate*, *stoppingCriterion* and *updateStepSize* are described in subsection 4.1.

space, at equal distance R from $\mathbf{S'_c}$. For each axis of the search space, the best solution among $\mathbf{S_{prev}}$, $\mathbf{S_{next}}$ and $\mathbf{S'_c}$ becomes the new candidate solution $\mathbf{S'_c}$. Other mechanisms can be used as that described in [11].

At the end of this procedure, the normalized direction vector $\mathbf{D'}$ from $\mathbf{S_c}$ to $\mathbf{S'_c}$, and the worst candidate solution $\mathbf{S_w}$, are also returned. $\mathbf{D'}$ is used later to update the step size, R, of the current agent, and $\mathbf{S_w}$ is used in the stopping criterion of the agent.

To illustrate this process, the successive displacement vectors of an agent, after its initialization, are presented in Figure 6 (a). Once normalized, these vectors are the direction vectors that are used to adapt the step size of the agent (denoted by $\mathbf{D}$ and $\mathbf{D'}$). As we can see, in Figure 6 (b), these displacement vectors are provided by the steps performed by the agent along each axis of the search space.

The adaptation of the step size R is performed through the procedure described in Figure 7. Depending on the situation, the step size is doubled or halved:

- if the procedure in Figure 5 cannot find a better candidate solution in the neighborhood of $\mathbf{S_c}$, *i.e.*, if $\mathbf{S'_c} = \mathbf{S_c}$, then R is halved;
- if the agent appears to be moving in a forward direction, according to the "cumulative dot product" described below, then R is doubled to accelerate the convergence of the agent.

The *cumulative dot product* makes use of trajectory information gathered along the steps of the agent. It is computed according to the equation (3) :

selectCandidate

Inputs: $\mathbf{S_c}, d, R$
local variables: $i, \mathbf{S_{prev}}, \mathbf{S_{next}}, \mathbf{S_i}$
$\mathbf{S'_c} \leftarrow \mathbf{S_c}$
$\mathbf{S_w} \leftarrow \mathbf{S_c}$
for $i = 1$ **to** d **do**
  $\mathbf{S_{prev}} \leftarrow \mathbf{S'_c} - R \times \mathbf{u}_i$
  $\mathbf{S_{next}} \leftarrow \mathbf{S'_c} + R \times \mathbf{u}_i$
  if $\mathbf{S_{prev}}$ is outside the search space **then**
    $\mathbf{S_{prev}} \leftarrow$ the closest point to $\mathbf{S_{prev}}$ inside the search space
  end
  if $\mathbf{S_{next}}$ is outside the search space **then**
    $\mathbf{S_{next}} \leftarrow$ the closest point to $\mathbf{S_{next}}$ inside the search space
  end
  Evaluate $\mathbf{S_{prev}}$ and $\mathbf{S_{next}}$
  $\mathbf{S_i} \leftarrow$ the best solution among $\mathbf{S_{prev}}$ and $\mathbf{S_{next}}$
  if $\mathbf{S_i}$ is strictly better than $\mathbf{S'_c}$ **then**
    $\mathbf{S'_c} \leftarrow \mathbf{S_i}$
  end
  $\mathbf{S_i} \leftarrow$ the worst solution among $\mathbf{S_{prev}}$ and $\mathbf{S_{next}}$
  if $\mathbf{S_i}$ is worse or equal to $\mathbf{S_w}$ **then**
    $\mathbf{S_w} \leftarrow \mathbf{S_i}$
  end
end
$\mathbf{D'} \leftarrow \mathbf{S'_c} - \mathbf{S_c}$
$\mathbf{D'} \leftarrow \frac{1}{\|\mathbf{D'}\|} \times \mathbf{D'}$
return $\{\mathbf{S'_c}, \mathbf{S_w}, \mathbf{D'}\}$

Fig. 5 Selection mechanism: selects the candidate solution $\mathbf{S'_c}$ to replace the current solution $\mathbf{S_c}$ of an agent. d is the dimension of the search space. R is the step size of the agent. $\mathbf{u}_i$ is the basis vector of the i^{th} axis of the search space. $\mathbf{S_w}$ is the worst tested candidate solution. $\mathbf{D'}$ is the direction vector of the current displacement of the agent. The function *Evaluate* computes the value of the objective function of a given solution and assigns this value to the solution.

$$U_n = \begin{cases} \frac{1}{2} \times U_{n-1} + \langle \mathbf{D}_{n-1}, \mathbf{D}_n \rangle & \text{if } n > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where U_n is the *cumulative dot product* of the successive direction vectors $\mathbf{D}_n$ of the displacements of the agent, at the step n since its initialization, and $\langle \mathbf{D}_{n-1}, \mathbf{D}_n \rangle$ is the dot product of the last two direction vectors. In the procedure *updateStepSize* (Figure 7), the sequence U_n corresponds to U, $\mathbf{D}_{n-1}$ corresponds to $\mathbf{D}$ and $\mathbf{D}_n$ corresponds to $\mathbf{D'}$ (the direction vectors of the previous and the current steps of the agent, respectively).

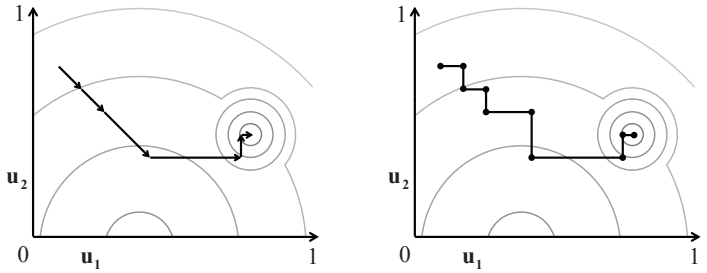

Fig. 6 Illustration of the local search performed by an agent on an example of a two-dimensional objective function. $\mathbf{u}_1$ and $\mathbf{u}_2$ are the normalized basis vectors of the search space. (a) shows the successive displacement vectors of the agent. Each of them is performed during one step of the local search procedure of the agent. (b) shows the path followed by the agent. This path is made of the best candidate solutions evaluated along each axis of the search space. These solutions are depicted by black-filled circles.

updateStepSize

Inputs: $\mathbf{S}'_\mathbf{c}, \mathbf{S}_\mathbf{c}, U, R, \mathbf{D}, \mathbf{D}', r_e$
local variables: $\varnothing$
if $\mathbf{S}'_\mathbf{c} = \mathbf{S}_\mathbf{c}$ **then**
  $U \leftarrow \frac{1}{2} \times U$
  $R \leftarrow \frac{1}{2} \times R$
else
  $U \leftarrow \frac{1}{2} \times U + \langle \mathbf{D}, \mathbf{D}' \rangle$
  if $U < -r_e$ **then**
    $U \leftarrow -r_e$
  else if $U > r_e$ **then**
    $R \leftarrow 2 \times R$
    if $R > 1$ **then**
      $R \leftarrow 1$
    end
    $U \leftarrow 0.0$
  end
end
return $\{U, R\}$

Fig. 7 Procedure to update the step size R of an agent. $\mathbf{S}'_\mathbf{c}$ is the candidate solution found to replace the current solution $\mathbf{S}_\mathbf{c}$ of the agent. U is the value of the *cumulative dot product* (see equation 3). $\mathbf{D}$ and $\mathbf{D}'$ are the direction vectors of the previous and the current displacements of the agent, respectively. r_e is a parameter of the algorithm.

In the case of U greater than r_e, R is doubled and U is reset to 0, where r_e is a parameter of the algorithm. To prevent U from having high negative values, at the end of the procedure *updateStepSize*, U is constrained to be higher or equal to $-r_e$.

```
stoppingCriterion
Inputs: S'_c, S_w, S_c, R, r_l, δ_ph, δ_pl
local variables: p
if R < r_l then
    if S'_c ≠ S_c then
        p ← |fitness(S'_c) − fitness(S_c)|
    else
        p ← |fitness(S_w) − fitness(S_c)|
    end
    if S_c is the best solution found since the last change in the objective function then
        if p ≤ δ_ph then
            return true
        end
    else
        if p ≤ δ_pl then
            return true
        end
    end
end
return false
```

Fig. 8 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. $\mathbf{S}'_{\mathbf{c}}$ is the candidate solution found to replace the current solution $\mathbf{S}_{\mathbf{c}}$ of the agent. $\mathbf{S}_{\mathbf{w}}$ 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.

The stopping criterion of the local search is presented in Figure 8, 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.

4.2 The diversity maintaining strategy

If an agent has found a local optimum, then it sends it to the coordinator that transmits it to the memory manager. Afterwards, the coordinator gives the agent its new position (the procedure used to generate an initial solution is detailed in subsection 4.3), in order to perform a new local search. This relocating is done by calling the procedure *initializeAgent* (see Figure 3) with the new position of the agent (and R is initialized to r_e).

relocateOrDestroyAgent

Inputs: N, n_a, d, A_m, A_i, r_e
local variables: $d_0, \mathbf{S_{new}}$
if $N > n_a$ **then**
  destroy the agent
end
$\{d_0, \mathbf{S_{new}}\} \leftarrow \text{newInitialSolution}(d, A_m, A_i)$
if $(N > 1)$ and $(d_0 \leq r_e)$ **then**
  destroy the agent
else
  relocate the agent at $\mathbf{S_{new}}$ with an initial step size of r_e
end

Fig. 9 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.

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.

4.3 The relocation of the agents

If an agent has found an optimum, then this optimum is transmitted to memory through the coordinator. Afterwards, the coordinator can either destroy the agent, or let the agent start a new local search at a given position. This decision is also taken for an agent that has been found too close to other agents (see Figure 2). This procedure is summarized in Figure 9.

We can note that this procedure makes use of two archives: A_m and A_i. The archive A_m contains the saved optima, and its capacity is equal to a fixed value n_m. We will see how this value is computed in subsection 4.5. 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 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 10). 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

```
newInitialSolution
Inputs: d, A_m, A_i
local variables: S, d_1, d_2, a_i, S_c, S_i
d_0 ← −∞
repeat 10 × d times
    S ← a random solution uniformly chosen in the search space
    d_1 ← +∞
    for each agent a_i do
        S_c ← the current solution of a_i
        d_2 ← the distance between S and S_c
        if d_2 < d_1 then
            d_1 ← d_2
        end
    end
    for each S_i ∈ (A_m ∪ A_i) do
        d_2 ← the distance between S and S_i
        if d_2 < d_1 then
            d_1 ← d_2
        end
    end
    if d_1 > d_0 then
        d_0 ← d_1
        S_new ← S
    end
end
return {d_0, S_new}
```

Fig. 10 Procedure to generate an initial solution $\mathbf{S_{new}}$ 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.

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 $\mathbf{S_{new}}$ 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 $\mathbf{S_{new}}$. 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 after the detection of a change in the objective function, as described in the next subsection.

addAgents

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

Fig. 11 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.

4.4 The change detection and the tracking of the optima

The coordinator detects the changes in the environment. 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 CHECK POINT state (see subsection 3.4). Changes in the environment 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 11) 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.

4.5 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:

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

where the *round* function rounds a number to the nearest integer, r_e is the exclusion radius of the agents and d is the dimension of the search space. This expression was defined empirically.

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 $\mathbf{O_c}$, 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 $\mathbf{O_c}$;
   (b) otherwise, the worst optimum of the archive is replaced by $\mathbf{O_c}$.
2. If there is one or several optima in the archive that are "too close" to $\mathbf{O_c}$ (an archived optimum is considered "too close" to $\mathbf{O_c}$ if it lies at a distance from $\mathbf{O_c}$ 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 12.

replaceDominatedOptima

Inputs: $\mathbf{O_c}, A_m, r_l, r_e$
local variables: $A_{sub}, \mathbf{O_{best}}, \mathbf{O_i}$
$A_{sub} \leftarrow \varnothing$
$\mathbf{O_{best}} \leftarrow \mathbf{O_c}$
for each $\mathbf{O_i} \in A_m$ **do**
  if $\|\mathbf{O_c} - \mathbf{O_i}\| \leq \sqrt{r_l \times r_e}$ **then**
    $A_{sub} \leftarrow A_{sub} \cup \{\mathbf{O_i}\}$
    if $\mathbf{O_i}$ is strictly better than $\mathbf{O_{best}}$ **then**
      $\mathbf{O_{best}} \leftarrow \mathbf{O_i}$
    end
  end
end
$A_m \leftarrow A_m - A_{sub}$
$A_m \leftarrow A_m \cup \{\mathbf{O_{best}}\}$
return A_m

Fig. 12 Procedure that replaces the dominated optima. $\mathbf{O_c}$ 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.

5 Benchmark sets

It is common that real-world problems have time costly evaluations of fitness functions. Hence, the computational cost of a tested algorithm in most testbeds is expressed in terms of number of evaluations. It is the case for the benchmarks used in this paper: the time corresponds to the number of evaluations since the beginning of a benchmark execution.

In dynamic environments, the goal is to quickly find and follow the global optimum over the time. Then, the measures used to evaluate the performance of dynamic optimization algorithms take into account this fact. In this section, the main two benchmarks in dynamic environments and their measures of performance are described.

5.1 The Moving Peaks Benchmark

To date, the most commonly used benchmark for dynamic optimization is the Moving Peaks Benchmark (MPB) [4, 5]. MPB is a maximization problem that consists of a number of peaks that randomly vary their shape, position and height upon time. At any time, one of the local optima can become the new global optimum. The peaks change position every α evaluations, and α is called "time span". They move by a fixed amount s (the change severity). More details about MPB are available in [5].

In order to evaluate the performance, the "offline error" is used. The offline error (oe) is defined in equation 5:

$$oe = \frac{1}{N_c} \sum_{j=1}^{N_c} \left(\frac{1}{N_e(j)} \sum_{i=1}^{N_e(j)} \left(f_j^* - f_{ji}^* \right) \right) \tag{5}$$

where N_c is the total number of fitness landscape changes within a single experiment, $N_e(j)$ is the number of evaluations 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.

In [5], 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 1), hence, it will be used in this paper.

5.2 The Generalized Dynamic Benchmark Generator

The Generalized Dynamic Benchmark Generator (GDBG) is the second benchmark used in this paper, it is described in [18, 20]. It was provided for the CEC'2009 Special Session on Evolutionary Computation in Dynamic and Uncertain Environments. It is based on the Sphere, Rastrigin, Weierstrass, Griewank and Ackley test functions

Parameter	Scenario 2
Number of peaks	10
Dimension d	5
Peak heights	$[30,70]$
Peak widths	$[1,12]$
Change cycle α	5000
Change severity s	1
Height severity	7
Width severity	1
Correlation coefficient	0
Number of changes N_c	100

Table 1 MPB parameters in scenario 2.

Parameter	Value
Dimension d (fixed)	10
Dimension d (changed)	$[5,15]$
Change cycle α	$10000 \times d$
Number of changes N_c	60

Table 2 GDBG parameters used during the CEC'2009 competition.

that are commonly used in the literature. They are presented in detail in [20]. These functions were rotated, composed and combined to form six problems with different degrees of difficulty:

F_1: rotation peak function (with 10 and 50 peaks)
F_2: composition of Sphere's function
F_3: composition of Rastrigin's function
F_4: composition of Griewank's function
F_5: composition of Ackley's function
F_6: hybrid composition function

A total of seven dynamic scenarios with different degrees of difficulty was proposed:

T_1: small step change (a small displacement)
T_2: large step change (a large displacement)
T_3: random change (Gaussian displacement)
T_4: chaotic change (logistic function)
T_5: recurrent change (a periodic displacement)
T_6: recurrent with noise (the same as above, but the optimum never returns exactly to the same point)
T_7: changing the dimension of the problem

The basic parameters of the benchmark are given in Table 2, where the change cycle corresponds, as for MPB, to the number of evaluations that makes a time span.

There are 49 test cases that correspond to the combinations of the six problems with the seven change scenarios (indeed, function F_1 is used twice, with 10 and 50

peaks respectively). For every test case, the tested algorithm is run several times. The number of runs of the tested algorithm is equal to 20 in our experiments.

As defined in [20], the convergence graphs, showing the relative error $r_i(t)$ of the run with median performance for each problem, are also computed. For the maximization problem F_1, the formula used for $r_i(t)$ is defined in equation 6, and for the minimization problems F_2 to F_6, it is defined in equation 7:

$$r_i(t) = \frac{f_i(t)}{f_i^*(t)} \tag{6}$$

$$r_i(t) = \frac{f_i^*(t)}{f_i(t)} \tag{7}$$

where $f_i(t)$ is the value of the best found solution at time t since the last occurrence of a change, during the i^{th} run of the tested algorithm, and $f_i^*(t)$ is the value of the global optimum at time t.

The marking scheme proposed by the authors of GDBG works as follows: a mark is calculated for each run of a test case, and the average value of this mark is denoted by $mark_{pct}$. The sum of all marks $mark_{pct}$ gives a score that corresponds to the overall performance of the tested algorithm, denoted by op. The percentage of the mark of each test case in this final score is defined by a coefficient $mark_{max}$. It is also the maximum value of $mark_{pct}$ that can be obtained by the tested algorithm on each test case. The authors of GDBG have set the values of the coefficients $mark_{max}$ such that the maximum value of op is equal to 100. This score is a measure of the performance of an algorithm in terms of both convergence speed and solution quality. It is based on the value of an approximated offline error, and on the value of the best relative error, calculated for each time span. More details about this marking scheme are given in [20].

6 Results and discussion

In the following subsections, the parameter setting of the proposed algorithm is discussed. Then, an analysis of the computational complexity of MLSDO is presented. Afterwards, its sensitivity analysis is given, followed by an empirical analysis of its components. A statistical analysis is performed to determine the strategies of MLSDO that are useful to improve its performance. Then, we show how the balance between exploring and tracking agents is dynamically adapted. Finally, a convergence analysis of MLSDO and a comparison with competing algorithms are presented.

6.1 Parameter setting of MLSDO

Table 3 summarizes the six parameters of MLSDO that the user has to define. In Table 3, the values given in the “MPB” column are suitable for the “Moving Peaks

Name	Type	Interval	MPB	GDBG	Short description
r_e	real	$(0,1]$	0.1	0.1	exclusion radius of the agents, and initial step size of "exploring" agents
r_l	real	$(0,r_e)$	0.005	0.005	initial step size of "tracking" agents
δ_{ph}	real	$[0,\delta_{pl}]$	0.001	0.001	highest precision parameter of the stagnation criterion of the agents local searches
δ_{pl}	real	$[\delta_{ph},+\infty]$	1.5	1.5	lowest precision parameter of the stagnation criterion of the agents local searches
n_a	integer	$[1,10]$	1	5	maximum number of "exploring" agents
n_c	integer	$[0,20]$	10	0	maximum number of "tracking" agents created after the detection of a change

Table 3 MLSDO parameter setting for MPB and GDBG.

Benchmark" (see subsection 5.1), and the ones given in the "GDBG" column are suitable for the "Generalized Dynamic Benchmark Generator" (see subsection 5.2). These values were fixed empirically, and used to perform all our experiments. Among several sets of values for the parameters, we selected the one that leads to the best performance.

The first parameters in Table 3 (r_e, r_l, δ_{ph} and δ_{pl}) that share the same values for the two benchmarks are suitable to induce a good performance both for MPB and GDBG. However, we do not have the proof that they are optimal for other dynamic problems.

The best performance on MPB (scenario 2) is obtained with $n_a = 1$. This can be explained by the characteristics of this scenario: the heights of the peaks are close during the first time span. Hence, there is no need to use several exploring agents in order to accelerate the discovery of a promising peak. Several agents would converge to local optima having a similar fitness, and the global convergence of the algorithm would be slowed down. However, having more than one exploring agent can lead to better performances for other problems, as it is the case for GDBG and for the problem used in subsection 6.3, based on the Rosenbrock function. Indeed, rather than performing sequentially a local search initialized in different areas of the search space, the use of several local searches performed in parallel can lead to a faster convergence to a good solution. For instance, if three agents are used in MLSDO, then at each iteration of the algorithm, each agent performs one step of its local search, *i.e.* one displacement from its current solution to a better one in its neighborhood (Figure 13 (c)). By contrast, if only one agent is used, sequentially, to perform these local searches, using the same initial solutions, then it has to complete one of these local searches before starting the next one (Figure 13 (a)). Hence, the parallel execution of the local searches can lead to a different convergence curve for MLSDO than the sequential one (Figures 13 (b) and (d)). If the fitness function is multimodal and composed of several peaks having different heights, then the average number of evaluations required to reach a good solution by the sequential execution of the local searches could be higher than by their parallel execution. It is especially the case if

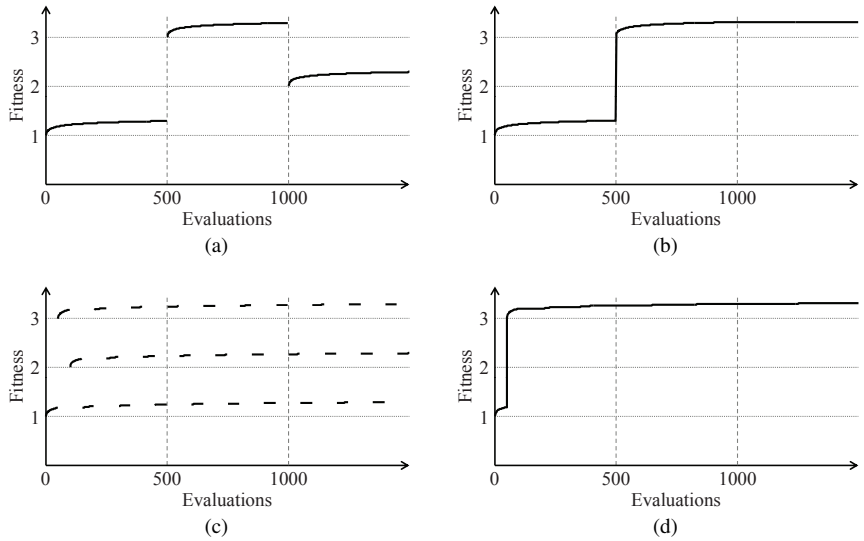

Fig. 13 Effects of the parallel and sequential execution of the local searches on the convergence of MLSDO. (a) illustrates a possible evolution of the fitness of the current solution over the evaluations, for three local searches performed sequentially. (b) illustrates the convergence curve that corresponds to this sequential execution. (c) illustrates the evolution of the fitness of the current solution for a parallel execution of these local searches. The same initial solutions are used, and the successive iterations of the local searches are just interleaved. (d) illustrates the convergence curve that corresponds to this parallel execution.

significant improvements are made early on in an agent's local search, with small improvements during final fine-tuning. This case is illustrated in Figure 13.
In this simple maximization example, we assume that:

- an agent needs 500 evaluations to complete its local search ;
- it needs 50 evaluations to perform one step of its local search ;
- the initial solutions of the local searches performed by the three agents are located in the attraction zone of different peaks having different heights ;
- the most significant improvements are made at the beginning of the local searches of the agents, with small improvements afterwards.

The offline performance is a performance measure defined as the average of $f^*(t)$, where $f^*(t)$ is the value of the best solution found at the t^{th} evaluation since the last change in the objective function [5]. The offline performance after 1500 evaluations, for each possible order by which the local searches are initialized, is presented in Table 4. As it can be seen, the parallel strategy leads to a better offline performance for 4 initialization orders among the 6 ones, and to a better average offline performance.

Then, we can say that depending on the width and height of the peaks, so depending on the nature of the problem, the parallel strategy may be better than the sequential one. For a problem with unknown characteristics, the parameter n_a can be fitted by testing its possible values from 1 to 10 and by keeping the best one.

	Initialization order						Average
	1,2,3	1,3,2	2,1,3	2,3,1	3,1,2	3,2,1	
Sequential	2.24	2.59	2.59	2.92	3.27	3.27	2.81
Parallel	3.14	3.18	3.17	3.21	3.25	3.25	3.20

Table 4 Offline performance for each possible order by which the initial solutions are taken to initialize the local searches (in the example illustrated in Figure 13, the initialization order is $2,1,3$). Its average value over all initialization orders is also given.

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 in a fast changing environment, 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 in fast changing environments.

As we can see in the procedure in Figure 11, the initial step size of "tracking" agents is equal to r_l, whereas the initial step size of "non-tracking" agents (the "exploring" ones) is equal to the exclusion radius r_e. 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. In this case, a low initial step size for "tracking" agents is needed to track an optimum because a larger initial step size will allow the agent to leave the attraction zone of the tracked optimum, and begin exploring the search space elsewhere. The parameter r_l has also to be well suited to the severity of the changes, since a too low value of r_l in a fast changing environment requires a significant number of adaptations of the step size, and a waste of many fitness function evaluations.

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.

For GDBG, we fixed $n_c = 0$. Thus, the archived local optima are not tracked but rather redetected using exploring agents. It means that the changes in the objective function are too strong for the use of tracking agents to be effective. Then, in order to achieve good performance for this benchmark, MLSDO has to quickly detect the new positions of the local optima, using exploring agents.

If the objective function is completely and randomly modified after a change, no information can be used from the past states of the function. Hence, it is not possible to do better than a random restart of the search process. Then, to be efficient in such environment, we need indeed a fast convergence to the global optimum, or at least, to a good solution.

Process	Complexity
relocation of agents which are exploring the same zone of the search space	$O(d^2(n_a+n_c)(n_a+n_c+n_m))$
execution of one step of the local search of each agent	$O(d(n_a+n_c))$
archiving the local optima found by the agents	$O(d\,n_m(n_a+n_c))$
detection of a change in the objective function, and reevaluation of the solutions of the agents and of the archives, if a change is detected	$O(n_a+n_c+n_m)$
creation of tracking agents, if a change is detected	$O(n_a+n_c)$
relocation of the agents having their stopping criterion satisfied	$O(d^2(n_a+n_c)(n_a+n_c+n_m))$
Total MLSDO complexity	$O(d^2(n_a+n_c)(n_a+n_c+n_m))$

Table 5 Computational complexity analysis of MLSDO.

6.2 Computational complexity analysis of MLSDO

All the procedures used in MLSDO are repeated between two changes in the objective function. Hence, the computational complexity of the algorithm is studied between two detections of a change. The processes performed between them are summarized in Table 5, with their computational complexity. The complexity of MLSDO, calculated by summing the complexities of these processes, is also given.

Therefore, for a fixed set of parameters of MLSDO, and by substituting the expression of n_m (equation (4)) into the expression of the complexity of MLSDO (in Table 5), we get a worst case computational complexity of $O(d^3)$ for MLSDO, where d is the dimension of the problem.

6.3 Sensitivity analysis of MLSDO

The sensitivity of MLSDO is studied by varying the value of one of its parameters, while the others are left unchanged. Each parameter is studied this way, by applying the algorithm on MPB (scenario 2) and on the Rosenbrock function in five dimensions (as defined in equation 8). The Rosenbrock function is used as a static minimization problem that admits only one global optimum equal to 0. We use this commonly used static test function in the analysis because the user may not know in advance if the problem to solve is static or dynamic. Hence, MLSDO should also be able to solve static functions, and the sensitivity of its parameters should also be studied for them.

$$f(\mathbf{x}) = \sum_{i=1}^{d-1} \left[(1-x_i)^2 + 100(x_{i+1}-x_i^2)^2\right] \tag{8}$$

where d is the number of dimensions, the search space is $[-10,10]^d$, $\mathbf{x}$ is a solution to be evaluated and x_i is its i^{th} coordinate.

The values used for the unchanged parameters are the ones defined in Table 3 for MPB, and the ones defined in Table 6 for the Rosenbrock function, *i.e.* the

Parameter	n_a	n_c	r_l	r_e	δ_{ph}	δ_{pl}
Value	2	0	0.005	0.03	1E−10	1E−04

Table 6 MLSDO parameter setting for the Rosenbrock function (used for the unchanged parameters in the sensitivity analysis).

ones fixed empirically that lead to the best performances. The ranges of values for the varying parameters have to be sufficiently wide for the analysis. We choose the following ones: $n_a \in [1,\ 10]$; $n_c \in [0,\ 20]$; $r_l \in [5\text{E}{-4},\ 1\text{E}{-2}]$; $r_e \in [0.01,\ 0.23]$ for both problems, $\delta_{ph} \in [3\text{E}{-8},\ 5\text{E}{-1}]$; $\delta_{pl} \in [1\text{E}{-3},\ 4\text{E}{+0}]$ for MPB, and $\delta_{ph} \in [5\text{E}{-11},\ 3\text{E}{-6}]$; $\delta_{pl} \in [3\text{E}{-6},\ 3\text{E}{-2}]$ for the Rosenbrock function.

Using MPB, the algorithm is stopped when 5×10^5 evaluations have been performed, and its performance is measured using the offline error (see subsection 5.1). Using the Rosenbrock function, it is stopped when the value of the best solution found by the algorithm falls below 0.01, and its performance is measured using the number of evaluations needed to satisfy this stopping criterion. We use this performance measurement on the Rosenbrock function because it is more important for a dynamic optimization algorithm to find the global optimum quickly than to find it accurately. For both test problems, the results are averaged over 100 runs, and illustrated in Figure 14 and Figure 15.

As we can see on both problems, when the number of agents n_a becomes too high, the performance of MLSDO decreases. Best results are obtained using $n_a = 1$ on MPB and using $n_a = 2$ on the Rosenbrock function. Varying the value of n_c does not perturb the performance of MLSDO on the Rosenbrock function, since it is a static function and thus, no tracking agent is used. However, on MPB, the performance of MLSDO increases with the value of n_c. Thus, n_c has to be high enough to let the algorithm track a sufficient number of local optima. It appears that a value greater or equal to 10 is fitted for MPB. On MPB, we can see that r_l, r_e and δ_{pl} have similar behavior, and that they can perturb the performance of MLSDO significantly if they are not correctly fitted. On the contrary, it appears that r_l has no impact on the performance of MLSDO on the Rosenbrock function, and that r_e and δ_{pl} do not highly perturb it. The behavior of δ_{ph} is similar in both test cases, and it needs to be sufficiently low to obtain accurate results. The optimal values of the MLSDO parameters are problem dependent, but their behaviors appear to be smooth and unimodal. Hence, the MLSDO parameters do not need to be accurately fitted in order to obtain good results. Besides, in these figures, we can see that the parameter values in Tables 3 and 6 correspond to the lowest values of offline error (Figure 14) and to the lowest numbers of evaluations (Figure 14), *i.e.* the values that correspond to the best performances for these benchmarks, according to these figures.

6.4 Empirical analysis of the strategies used in MLSDO

An evaluation of the components of the MLSDO algorithm is made, in order to justify their requirement for obtaining high quality results. This evaluation is performed on

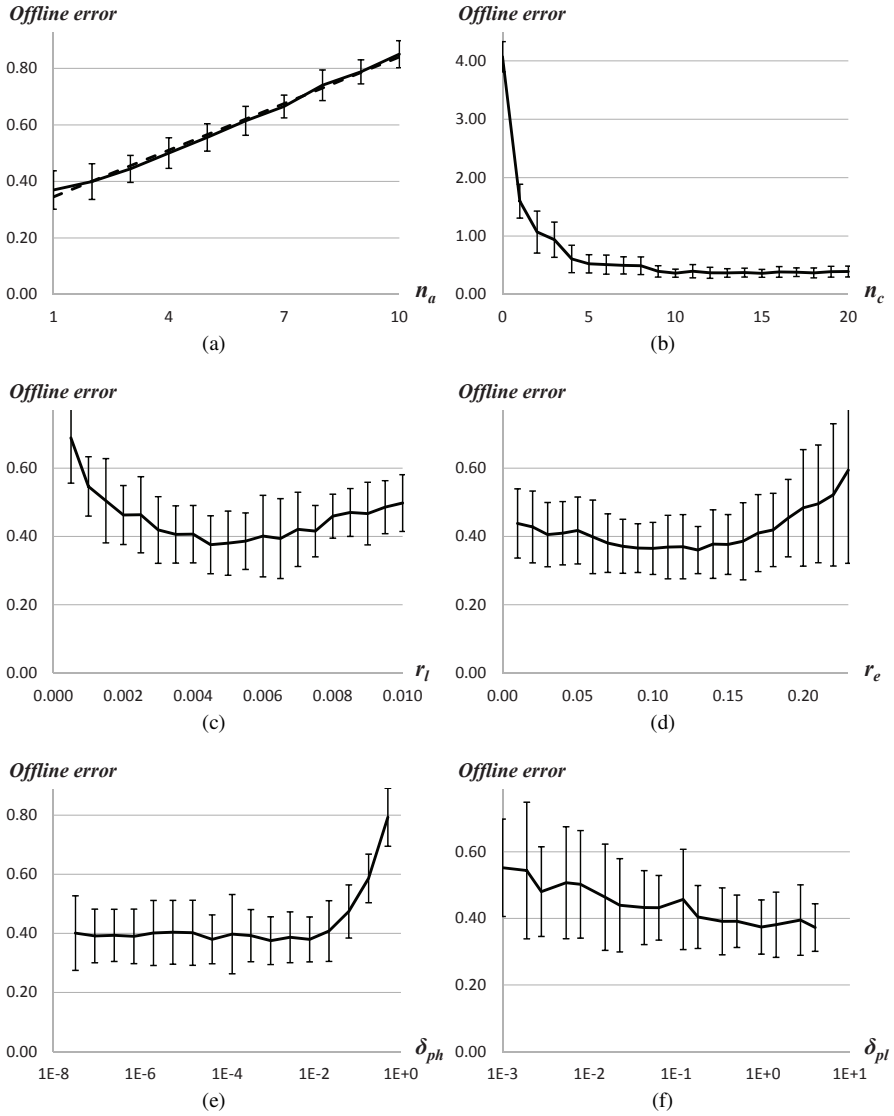

Fig. 14 Sensitivity on MPB. (a) represents the evolution of the offline error for several values of the parameter n_a. (b) represents it for n_c. (c) represents it for r_l. (d) represents it for r_e. (e) represents it for δ_{ph}. (f) represents it for δ_{pl}.

MPB (scenario 2, because it is the most used one among the three scenarios proposed in [5], see subsection 5.1) and on the problems F_2 and F_3 of the GDBG benchmark (based on the unimodal Sphere function and on the multimodal Rastrigin function, respectively). Using MPB, the maximum number of iterations is fixed to 5×10^5. It corresponds to 100 changes occurred in the objective function during each run. The resulting offline errors and standard deviations averaged over 100 runs are summa-

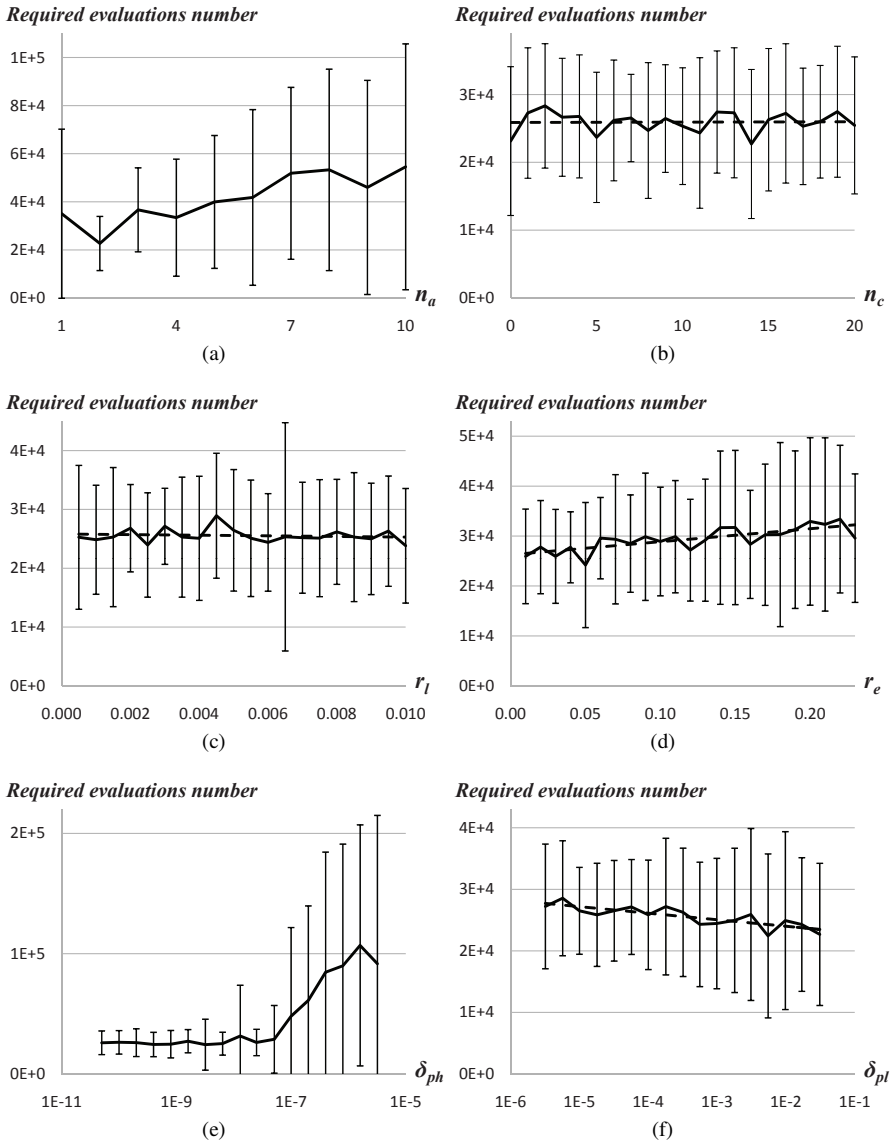

Fig. 15 Sensitivity on the Rosenbrock function. (a) represents the evolution of the number of evaluations needed to find the global optimum (with a precision of 0.01) for several values of the parameter n_a. (b) represents it for n_c. (c) represents it for r_l. (d) represents it for r_e. (e) represents it for δ_{ph}. (f) represents it for δ_{pl}.

rized in the second column of Table 7, for several variants of MLSDO. Using F_2 and F_3, the sum of the marks $mark_{run}$, obtained for the seven change scenarios of GDBG, gives a score for each variant of MLSDO. These scores and their standard deviations averaged over 20 runs are summarized in the third and fourth columns of Table 7. In

Variant	*oe* for MPB	Score for F_2	Score for F_3	Short description
$MLSDO_{noCDPA}$	0.35 ± 0.09	14.30 ± 0.19	$\mathbf{3.41 \pm 0.20}$	no cumulative dot product adaptation of R
MLSDO	0.36 ± 0.08	13.93 ± 0.21	6.74 ± 0.20	the unmodified MLSDO algorithm
$MLSDO_{noExcl}$	0.42 ± 0.11	$\mathbf{13.51 \pm 0.24}$	6.74 ± 0.21	no exclusion radius for the agents
$MLSDO_{no\,\delta_{pl}}$	$\mathbf{0.59 \pm 0.26}$	13.93 ± 0.22	$\mathbf{5.88 \pm 0.21}$	using only δ_{ph} in the stopping criterion
$MLSDO_{randInit}$	$\mathbf{0.61 \pm 0.24}$	13.89 ± 0.23	6.74 ± 0.29	with uniform random initial solutions for the local searches of the agents
$MLSDO_{no\,r_l}$	$\mathbf{0.85 \pm 0.09}$	13.89 ± 0.23	6.68 ± 0.23	using r_e instead of r_l as initial step size of "tracking" agents
$MLSDO_{no\,\delta_{ph}}$	$\mathbf{1.08 \pm 0.07}$	$\mathbf{12.46 \pm 0.26}$	$\mathbf{6.01 \pm 0.21}$	using only δ_{pl} in the stopping criterion
$MLSDO_{noDom}$	$\mathbf{1.34 \pm 0.44}$	13.83 ± 0.22	6.70 ± 0.17	without removing dominated optima from the archive of local optima
$MLSDO_{noTrack}$	$\mathbf{4.08 \pm 0.25}$	13.93 ± 0.25	6.73 ± 0.21	without creating "tracking" agents
$MLSDO_{noArch}$	$\mathbf{7.22 \pm 0.34}$	13.84 ± 0.23	6.76 ± 0.25	without archiving local optima

Table 7 Performance of each simplified variant of MLSDO for MPB and the problems F_2 and F_3 of GDBG.

this table, the variants are sorted from the best to the worst according to the offline error (*oe*) obtained for MPB.

The Kruskal-Wallis statistical test has been applied on the results obtained by the variants of MLSDO. For MPB, as well as for the problems F_2 and F_3, this test indicates at 99% confidence level that there is a significant difference between the performances of at least two variants. Then, the Tukey-Kramer multiple comparisons procedure has been used to determine which variants differ in terms of offline error for MPB, and in terms of score for F_2 and F_3. The results obtained by the variants that perform significantly differently from the unmodified MLSDO algorithm, at 99% confidence level, appear in bold in this table.

Among the simpler variants of MLSDO, we can note that $MLSDO_{noExcl}$ 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 local search elsewhere when it is too close to another agent. Thus, all agents can explore the same zone of the search space. In $MLSDO_{randInit}$, the heuristic in Figure 10 is not used. This heuristic generates an initial solution for an agent that is far from the already explored zones of the search space. Hence, in $MLSDO_{randInit}$, the initial solutions of the local searches of the agents are randomly generated uniformly in the search space. In $MLSDO_{noCDPA}$, 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). In $MLSDO_{no\,r_l}$, the initial step size of a "tracking"

agent, created by the coordinator when a change is detected in the objective function (on the location of a previously found optimum), is not equal to r_l but to r_e. Hence, the initial step size of a "tracking" agent is the same as the one of an "exploring" agent, in this variant. In MLSDO$_{\text{noArch}}$, the local optima found by the agents are not stored (A_m is always empty). In MLSDO$_{\text{noDom}}$, the procedure in Figure 12 is not used. Thus, the local optima that are dominated by another one are not removed from A_m. In MLSDO$_{\text{no}\,\delta_{pl}}$, δ_{pl} is replaced by δ_{ph}, so that the precision of the stopping criterion is always equal to δ_{ph} (the parameter δ_{pl} is not used in this variant). The opposite replacement is done in MLSDO$_{\text{no}\,\delta_{ph}}$, where the parameter δ_{ph} is not used. Finally, we can also note that in MLSDO$_{\text{noTrack}}$, no "tracking" agent is created when a change is detected in the objective function (the parameter n_c is not used).

In Table 7, the results obtained by MLSDO$_{\text{noArch}}$ and MLSDO$_{\text{noTrack}}$ are far worse than the ones of the other variants on MPB. Hence, we can conclude that the most important components of the MLSDO algorithm, in order to achieve a better result, are the archiving of found local optima and their tracking using dedicated agents. Besides, we can see that MLSDO$_{\text{noArch}}$ performs significantly worse than MLSDO$_{\text{noTrack}}$. Indeed, in MLSDO$_{\text{noArch}}$, the found local optima are not archived. Thus, they can neither be tracked, nor be used by the heuristic in Figure 10 to generate initial solutions for the agents that are far from these found local optima. However, in MLSDO$_{\text{noTrack}}$, the local optima are not tracked, but they are used in the generation of the initial solutions of the agents. Thus, we can also conclude that the use of A_m in this generation process is important to achieve a good performance.

By comparing the results obtained by MLSDO$_{\text{no}\,\delta_{pl}}$ and MLSDO$_{\text{no}\,\delta_{ph}}$, we can conclude that the use of the lowest precision parameter is less critical than the highest one, in order to achieve a better result. Hence, if we have only one of these parameters in the stopping criterion, it is better to set it to a low value in order to locate all the local optima with a high precision. However, the combined use of these two levels of precision is required to significantly improve the performance of MLSDO.

Looking at the results obtained by MLSDO$_{\text{randInit}}$, MLSDO$_{\text{no}\,r_l}$ and MLSDO$_{\text{noDom}}$, we can also conclude that an intelligent generation of the initial solutions for the agents, as well as the use of a dedicated initial step size r_l for the "tracking" agents, and the removal of dominated optima from the archive of local optima, are important strategies in MLSDO.

Finally, the statistical analysis shows that a significant difference exists between the unmodified MLSDO algorithm and all its variants except MLSDO$_{\text{noCDPA}}$ and MLSDO$_{\text{noExcl}}$ on MPB, at 99% confidence level as well as at 95%. Hence, we can conclude that all the tested components of MLSDO, except the cumulative dot product adaptation of the step size of an agent and the use of an exclusion radius, are useful to get good results on MPB. However, MLSDO performs significantly better than MLSDO$_{\text{noCDPA}}$ for the problem F_3 of GDBG, and significantly better than MLSDO$_{\text{noExcl}}$ for the problem F_2. Hence, even if these strategies are not helpful for MPB, they can greatly improve the performance of MLSDO for other dynamic problems.

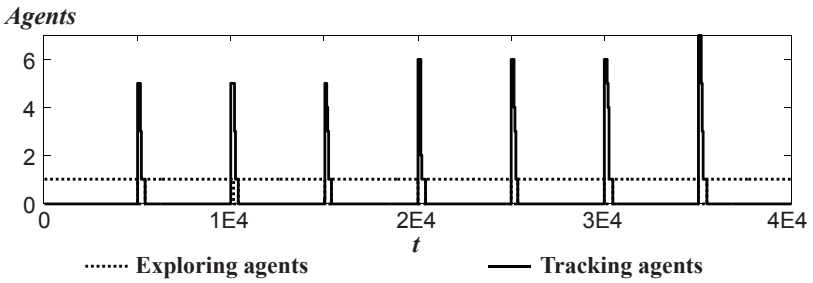

Fig. 16 Evolution of the number of exploring and tracking agents during the first 40000 evaluations of a run of MPB.

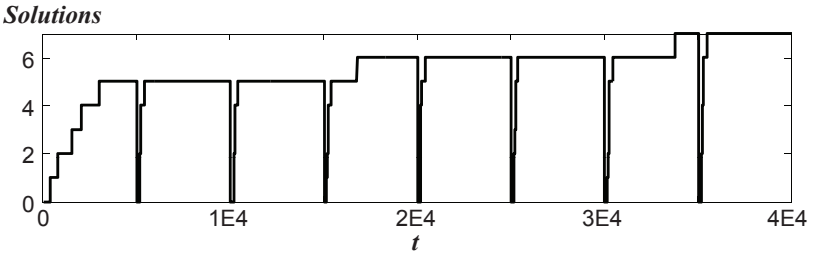

Fig. 17 Evolution of the number of solutions stored in the archive of the found local optima, during the first 40000 evaluations of a run of MPB.

6.5 Analysis of the balance between exploring and tracking agents

The evolution of the number of exploring and tracking agents over the time is shown in Figure 16, for the first 40000 evaluations of a run of MLSDO on MPB (scenario 2). The evolution of the number of archived local optima is also shown in Figure 17, as it is linked to the number of tracking agents created after a change, *i.e.* the highest number of tracking agents created after a change is equal to the number of archived optima.

As we can see, during the first time span, five local optima are found among ten. Then, the remaining ones are progressively detected. The tracking agents require few evaluations to locate the new position of the detected local optima. Hence, most evaluations of a time span of MPB are used to explore the search space, in order to find the local optima that remain to be detected.

For the GDBG benchmark, the number of exploring agents stays equal to five, and no tracking agent is created during a run. Indeed, we only make use of exploring agents to detect the new position of the local optima.

6.6 Convergence analysis and comparison on MPB

The convergence of MLSDO is studied in Figure 18, using the run with median performance among 100 runs on MPB. As one can see, the convergence of MLSDO on each time span is fast. The first time spans show the highest values, because MLSDO

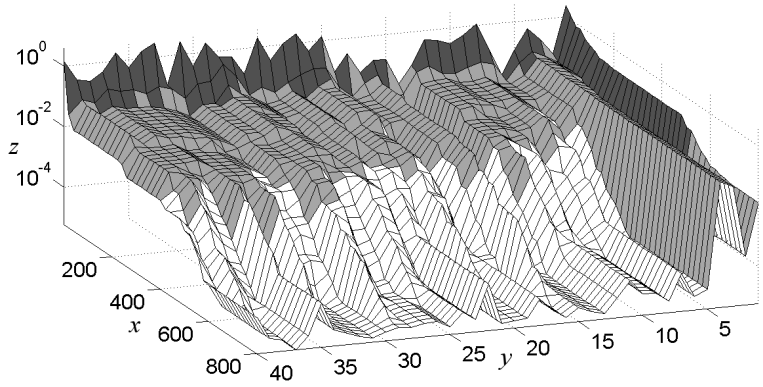

Fig. 18 Relative error of the fitness on MPB. The axis y corresponds to the first 40 time spans, the axis x corresponds to the first 800 evaluations of a time span with a granularity of 20 evaluations, and the axis z is equal to the relative error $\frac{f^*_y - f^*_{yx}}{f^*_y}$, using the notations of equation 5. For more clarity, we used a logarithmic scale for the axis z.

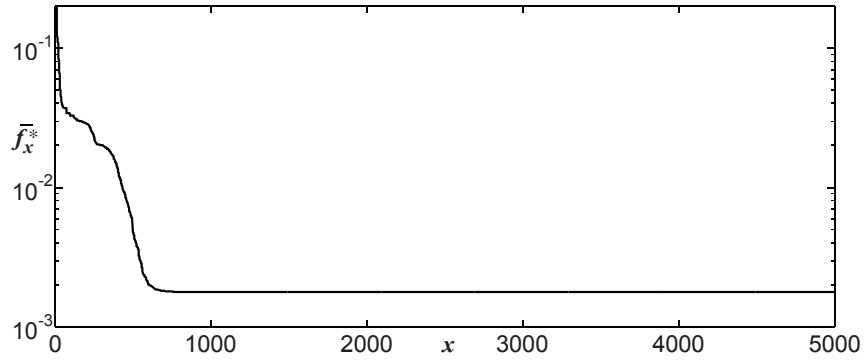

Fig. 19 Average relative error of the fitness on MPB. x corresponds to the evaluations of a time span, and $\bar{f}^*_x$ is the average value of f^*_{jx} for $j = 1,...,N_c$. For more clarity, we used a logarithmic scale for $\bar{f}^*_x$ axis.

has not yet recorded the locations of each local optimum of the landscape. Once 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 19. As illustrated in this figure, it takes only 20 evaluations to get a relative error lower than 10^{-1}, and 431 evaluations to get a relative error lower than 10^{-2}.

The comparison, on MPB, of MLSDO with the other leading optimization algorithms in dynamic environments is summarized in Table 8. These competing algorithms are the only ones that we found suitable for comparison in the literature, *i.e.*, they are tested by their authors using the most commonly used set of MPB parameters (see Table 1). The offline errors and the standard deviations are given, and the algorithms are sorted in increasing order of offline error. Results are averaged over 50 runs of the tested algorithms. Results of competing algorithms are given in the

Algorithm	Offline error
Moser and Chiong, 2010 [27]	0.25 ± 0.08
MLSDO	0.36 ± 0.08
Novoa *et al.*, 2009 [29]	0.40 ± 0.04
Lepagnot *et al.*, 2009 [16, 17]	0.59 ± 0.10
Moser and Hendtlass, 2007 [27, 28]	0.66 ± 0.20
Yang and Li, 2010 [36]	1.06 ± 0.24
Liu *et al.*, 2010 [23]	1.31 ± 0.06
Lung and Dumitrescu, 2007 [24]	1.38 ± 0.02
Bird and Li, 2007 [1]	1.50 ± 0.08
Lung and Dumitrescu, 2008 [25]	1.53 ± 0.01
Blackwell and Branke, 2006 [3]	1.72 ± 0.06
Mendes and Mohais, 2005 [26]	1.75 ± 0.03
Li *et al.*, 2006 [22]	1.93 ± 0.06
Blackwell and Branke, 2004 [2]	2.16 ± 0.06
Parrott and Li, 2006 [31]	2.51 ± 0.09
Du and Li, 2008 [9]	4.02 ± 0.56

Table 8 Comparison with competing algorithms on MPB using standard settings (see Table 1).

Algorithm	Offline error
MLSDO	14.00 ± 2.33
Moser and Chiong, 2010 [27]	16.50 ± 5.40
Lung and Dumitrescu, 2007 [24]	24.60 ± 0.25
Lepagnot *et al.*, 2009 [16, 17]	34.64 ± 2.72
Moser and Hendtlass, 2007 [27, 28]	480.5 ± 70.1

Table 9 Comparison with competing algorithms on MPB using $d = 100$.

references listed in the first column. As we can see, MLSDO is the second ranked algorithm in terms of offline error. The first ranked algorithm on MPB is the one proposed by Moser and Chiong in 2010, called Hybridised EO [27].

Though they differ in several points, both Hybridised EO and MLSDO restart local searches repeatedly in the search space, and store the found local optima in order to track them after a change. Compared to MLSDO, Hybridised EO is a "multi-phase" algorithm that does not perform several local searches in parallel. An interesting difference between them is that, in order to generate an initial solution for a local search in a promising area of the search space, Hybridised EO evaluates several candidate solutions and uses the best one as the initial solution of the local search. More precisely, it samples every dimension of the search space in equal distances. Thus, in order to start a local search in a d-dimensional space, it has first to evaluate $10 \times d$ candidate solutions. Then, the question of the scalability of the algorithm regarding the number of dimensions of the problem arises. In comparison, MLSDO uses only the archived solutions to produce an initial solution for a local search, *i.e.* it does not evaluate any additional one.

Then, to compare the performance of MLSDO with the other competing algorithms in the high dimensional case, the same comparison (on MPB, scenario 2) is made using $d = 100$, as summarized in Table 9. However, the numerical results using $d = 100$ of several algorithms of Table 8 are not available (they are not included in this comparison). The results of the algorithm proposed by Lung and Dumitrescu [24] are

Algorithm	Offline error		
	$s = 1$	$s = 2$	$s = 3$
Moser and Chiong, 2010 [27]	0.25 ± 0.08	0.47 ± 0.12	0.49 ± 0.12
MLSDO	0.36 ± 0.08	0.60 ± 0.07	0.92 ± 0.10
Lepagnot *et al.*, 2009 [16, 17]	0.59 ± 0.10	0.87 ± 0.12	1.18 ± 0.13
Moser and Hendtlass, 2007 [27, 28]	0.66 ± 0.20	0.86 ± 0.21	0.94 ± 0.22
Yang and Li, 2010 [36]	1.06 ± 0.24	1.17 ± 0.22	1.36 ± 0.28
Liu *et al.*, 2010 [23]	1.31 ± 0.06	1.98 ± 0.06	2.21 ± 0.06
Lung and Dumitrescu, 2007 [24]	1.38 ± 0.02	1.78 ± 0.02	2.03 ± 0.03
Bird and Li, 2007 [1]	1.50 ± 0.08	1.87 ± 0.05	2.40 ± 0.08
Lung and Dumitrescu, 2008 [25]	1.53 ± 0.01	1.57 ± 0.01	1.67 ± 0.01
Blackwell and Branke, 2006 [3]	1.75 ± 0.06	2.40 ± 0.06	3.00 ± 0.06
Li *et al.*, 2006 [22]	1.93 ± 0.08	2.25 ± 0.09	2.74 ± 0.09
Parrott and Li, 2006 [31]	2.51 ± 0.09	3.78 ± 0.09	4.96 ± 0.12

Table 10 Comparison with competing algorithms on MPB using $s = 1, 2, 3$.

taken from [27]. The results of the remaining competing algorithms are taken from their corresponding published papers.

As we can see, MLSDO is the best ranked algorithm in terms of offline error, in this high dimensional case, whereas it was ranked at the second place using $d = 5$, *i.e.*, MLSDO obtains a worse offline error than the algorithm proposed by Moser and Chiong [27] using $d = 5$, whereas it is the opposite using $d = 100$. It means that the performance of MLSDO scales well regarding the number of dimensions of the problem, compared to competing algorithms.

Finally, we compare the performance on MPB (scenario 2) of MLSDO with the competing algorithms that provide comparable numerical results, using higher change severities. It is made using $s = 1, 2, ..., 6$, as summarized in Tables 10 and 11. The results of the algorithm proposed by Li *et al.* [22] are from [23], and the results of the algorithm proposed by Parrott and Li [31] are from [1]. The results of the remaining competing algorithms are taken from their corresponding papers.

It is known that the optima are more and more difficult to track with the increase of the change severity. Therefore, the performance of all algorithms degrades when the change severity increases. However, we can see that the performance of the following four algorithms does not degrade as much as the one of MLSDO: Moser and Chiong, 2010 [27]; Moser and Hendtlass, 2007 [27, 28]; Yang and Li, 2010 [36] and Lung and Dumitrescu, 2008 [25]. Hence, though MLSDO is ranked at the second place using $s = 1, 2, 3$, it is ranked at the third, fourth and fifth place for $s = 4, 5, 6$, respectively.

This decrease in the performance of MLSDO is explained by the fact that the initial step size r_l of the tracking agents has to be adapted to the severity of the changes, as stated in subsection 6.1. Therefore, better results can be obtained by adapting the value of r_l to the change severity s, as shown in Table 12.

Algorithm	Offline error		
	$s=4$	$s=5$	$s=6$
Moser and Chiong, 2010 [27]	0.53 ± 0.13	0.65 ± 0.19	0.77 ± 0.24
MLSDO	1.22 ± 0.12	1.62 ± 0.13	2.01 ± 0.19
Lepagnot *et al.*, 2009 [16, 17]	1.49 ± 0.13	1.86 ± 0.17	2.32 ± 0.18
Moser and Hendtlass, 2007 [27, 28]	0.97 ± 0.21	1.05 ± 0.21	1.09 ± 0.22
Yang and Li, 2010 [36]	1.38 ± 0.29	1.58 ± 0.32	1.53 ± 0.29
Liu *et al.*, 2010 [23]	2.61 ± 0.11	3.20 ± 0.13	3.93 ± 0.14
Lung and Dumitrescu, 2007 [24]	2.23 ± 0.05	2.52 ± 0.06	2.74 ± 0.10
Bird and Li, 2007 [1]	2.90 ± 0.08	3.25 ± 0.09	3.86 ± 0.11
Lung and Dumitrescu, 2008 [25]	1.72 ± 0.03	1.78 ± 0.06	1.79 ± 0.03
Blackwell and Branke, 2006 [3]	3.59 ± 0.10	4.24 ± 0.10	4.79 ± 0.10
Li *et al.*, 2006 [22]	3.05 ± 0.10	3.24 ± 0.11	4.95 ± 0.13
Parrott and Li, 2006 [31]	5.56 ± 0.13	6.76 ± 0.15	7.68 ± 0.16

Table 11 Comparison with competing algorithms on MPB using $s = 4,5,6$.

Value of s	1	2	3	4	5	6
Value of r_l	0.005	0.010	0.015	0.019	0.023	0.027
Offline error	0.36 ± 0.08	0.54 ± 0.09	0.67 ± 0.09	0.84 ± 0.08	1.00 ± 0.12	1.38 ± 0.14

Table 12 Performance of MLSDO on MPB using $s = 1,2,...,6$. The parameter r_l of MLSDO is empirically adapted to the value of s used. The other parameters of MLSDO are left unchanged to the values defined in subsection 6.1.

6.7 Convergence analysis and comparison on GDBG

The experimental results on the GDBG benchmark are gathered in Table 13. An example of the convergence behavior of MLSDO on the problem F_1 with 10 peaks is depicted in Figure 20, where t is the number of evaluations since the beginning of the run. In this figure, the relative error $r_i(t)$ is given for each change scenario T_k as $r_i(t) + k - 1$, where $0 \leq r_i(t) \leq 1$, i is the run with median performance and $k = 1,2,...,7$.

As we can see, the curves of the relative error for all change scenarios are sharp. It means that MLSDO needs only few evaluations to converge to a better local optimum. To illustrate this fast convergence, a zoom on the first change in the objective function of this problem, for all change scenarios, is made on Figure 20.

On the change scenarios T_1, T_4 and T_6, the relative error reaches a value close to 1 during each time span. It means that the global optimum is found for all the time spans of these three scenarios. The average number of evaluations required to reach a relative error of 0.99 during a time span is equal to 2511 on T_1, 1596 on T_4 and 5523 on T_6. Thus, MLSDO provides a fast convergence to the global optimum on these change scenarios. On the other scenarios, we can see that the global optimum is also found quickly for most time spans.

As we can see in Table 13, MLSDO is able to produce reasonable solutions for all problems. Only for problem F_3, the proposed MLSDO algorithm could not find the optimum quick enough during dynamic changes. This is however a hard problem, and all competing algorithms have difficulties to solve it. The best mark obtained by

Function	Change	mark_{max}	mark_{pct}
F_1 (10 peaks)	T_1	1.5	1.493
F_1 (10 peaks)	T_2	1.5	1.455
F_1 (10 peaks)	T_3	1.5	1.428
F_1 (10 peaks)	T_4	1.5	1.496
F_1 (10 peaks)	T_5	1.5	1.464
F_1 (10 peaks)	T_6	1.5	1.481
F_1 (10 peaks)	T_7	1.0	0.966
Sum for F_1 (10 peaks)		10.0	9.78
F_1 (50 peaks)	T_1	1.5	1.490
F_1 (50 peaks)	T_2	1.5	1.433
F_1 (50 peaks)	T_3	1.5	1.386
F_1 (50 peaks)	T_4	1.5	1.481
F_1 (50 peaks)	T_5	1.5	1.469
F_1 (50 peaks)	T_6	1.5	1.473
F_1 (50 peaks)	T_7	1.0	0.934
Sum for F_1 (50 peaks)		10.0	9.67
F_2	T_1	2.4	2.277
F_2	T_2	2.4	1.876
F_2	T_3	2.4	1.940
F_2	T_4	2.4	2.348
F_2	T_5	2.4	1.762
F_2	T_6	2.4	2.258
F_2	T_7	1.6	1.469
Sum for F_2		16.0	13.93
F_3	T_1	2.4	1.913
F_3	T_2	2.4	0.461
F_3	T_3	2.4	0.821
F_3	T_4	2.4	1.250
F_3	T_5	2.4	0.652
F_3	T_6	2.4	0.920
F_3	T_7	1.6	0.724
Sum for F_3		16.0	6.74
F_4	T_1	2.4	2.261
F_4	T_2	2.4	1.808
F_4	T_3	2.4	1.863
F_4	T_4	2.4	2.273
F_4	T_5	2.4	1.681
F_4	T_6	2.4	2.226
F_4	T_7	1.6	1.427
Sum for F_4		16.0	13.54
F_5	T_1	2.4	2.280
F_5	T_2	2.4	2.276
F_5	T_3	2.4	2.278
F_5	T_4	2.4	2.288
F_5	T_5	2.4	2.268
F_5	T_6	2.4	2.272
F_5	T_7	1.6	1.520
Sum for F_5		16.0	15.18
F_6	T_1	2.4	1.945
F_6	T_2	2.4	1.792
F_6	T_3	2.4	1.763
F_6	T_4	2.4	1.891
F_6	T_5	2.4	1.935
F_6	T_6	2.4	1.832
F_6	T_7	1.6	1.277
Sum for F_6		16.0	12.44
Overall performance op		100.0	**81.28**

Table 13 Performance measurement on GDBG.

competing algorithms on the problem F_3 is 5.15 [7], and MLSDO obtains a better score of 6.74.

Among the 49 test cases summarized in Table 13, MLSDO performs worst for the problem F_3 using the change scenario T_2. It can be explained by the large displacements of the optima produced by this change scenario, inducing a major modification of the landscape of this difficult problem.

MLSDO needs only few evaluations per *time span* to find a good solution. This is a major advantage in a fast changing environment.

The results obtained by the competing algorithms on GDBG are summarized in Tables 14 and 15. For each problem of GDBG, the sum of $mark_{pct}$ obtained for the seven change scenarios is given in Table 14. To make the results obtained by the algorithms comparable, we use the same values of $mark_{max}$ for the problem F_1 as for the other problems, *i.e.* for each problem, $mark_{max} = 1.6$ for the change scenario T_7 and $mark_{max} = 2.4$ for the other change scenarios.

Conversely, for each change scenario, the sum of $mark_{pct}$ obtained for all problems is given in Table 15. We use the same values of $mark_{max}$ for the change scenario

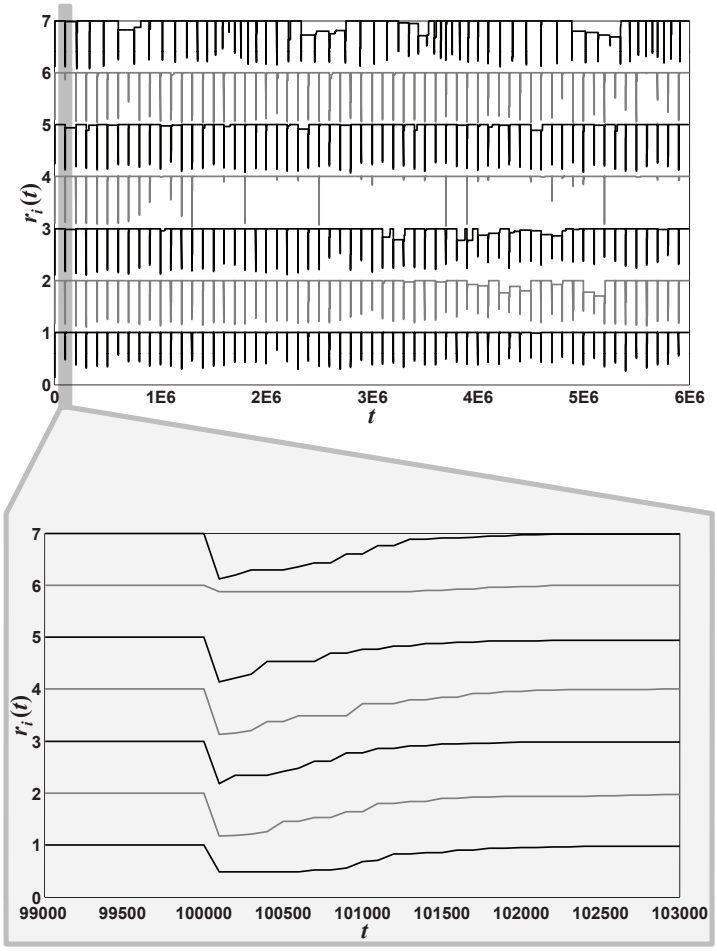

Fig. 20 Convergence graph for the GDBG problem F_1 (10 peaks).

T_7 as for the other scenarios, *i.e.* for each change scenario, $mark_{max} = 1.5$ for the problem F_1 (with 10 and 50 peaks) and $mark_{max} = 2.4$ for the other problems.

From Table 14, we can see that every algorithm performs best for the function F_1, and worst for the functions F_3 and F_6, though the results obtained for F_6 are better than for F_3. For the other functions, results are mitigated and we cannot conclude. Compared to the other algorithms, MLSDO obtains the best results for all functions except F_2 and F_4.

From Table 15, we can see that the algorithms perform best for the change scenarios T_1 and T_4. For the other scenarios, results are mitigated and we cannot conclude. Compared to the other algorithms, MLSDO obtains the best results for all change scenarios.

Algorithm	Performance						
	F_1 (10 peaks)	F_1 (50 peaks)	F_2	F_3	F_4	F_5	F_6
MLSDO	15.65	15.47	13.93	6.74	13.54	15.18	12.44
Lepagnot *et al.*, 2009 [16, 17]	15.64	15.44	14.65	0.20	14.12	13.26	9.10
Brest *et al.*, 2009 [7]	15.07	15.00	10.91	5.15	10.78	14.16	9.94
Korosec and Silc, 2009 [15]	14.66	14.52	11.25	3.36	9.90	13.50	8.96
Yu and Suganthan, 2009 [38]	13.13	13.30	10.64	4.37	10.31	8.95	7.31
Li and Yang, 2009 [19]	14.45	14.36	10.64	1.70	9.47	10.30	7.46
França and Zuben, 2009 [10]	12.22	12.63	8.55	0.16	6.72	4.17	3.15

Table 14 Performances of the competing algorithms for each problem of GDBG. In this table, the performance of an algorithm for a problem is calculated as the sum of $mark_{pct}$ obtained for the seven change scenarios, using $mark_{max} = 1.6$ for T_7 and $mark_{max} = 2.4$ for the other change scenarios.

Algorithm	Performance						
	T_1	T_2	T_3	T_4	T_5	T_6	T_7
MLSDO	13.66	11.10	11.48	13.03	11.23	12.46	12.47
Lepagnot *et al.*, 2009 [16, 17]	11.28	10.42	10.19	10.97	10.20	10.66	10.56
Brest *et al.*, 2009 [7]	12.57	9.18	9.19	12.06	9.43	10.58	10.08
Korosec and Silc, 2009 [15]	11.46	8.61	8.97	10.65	8.95	10.02	9.80
Yu and Suganthan, 2009 [38]	8.58	8.11	8.42	10.10	8.53	8.89	8.19
Li and Yang, 2009 [19]	9.78	7.75	7.70	11.49	7.71	7.78	8.06
França and Zuben, 2009 [10]	8.12	5.45	5.57	6.89	3.55	4.96	5.63

Table 15 Performances of the competing algorithms for each change scenario of GDBG. In this table, the performance of an algorithm for a change scenario is calculated as the sum of $mark_{pct}$ obtained for all problems, using $mark_{max} = 1.5$ for F_1 and $mark_{max} = 2.4$ for the other problems.

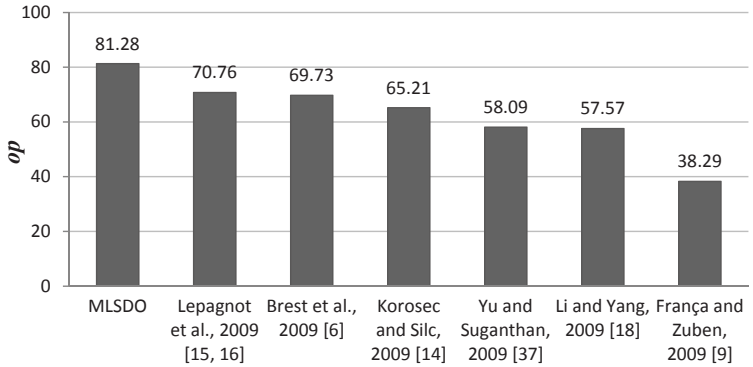

Fig. 21 Comparison with competing algorithms on GDBG.

The comparison, on GDBG, of MLSDO with the other leading optimization algorithms in dynamic environments is summarized in Figure 21. The algorithms are ranked according to their overall performance. As we can see, MLSDO is the first ranked algorithm on this benchmark.

7 Conclusion

A new algorithm for DOPs has been proposed, called MLSDO. It has been developed in order to solve a wide range of DOPs. It is a cooperative search algorithm based on several coordinated local search agents, and on the archiving of the found local optima, in order to track them after a change in the objective function. It makes use of the main strategies and cooperative techniques proposed for DOPs in the literature, and implements them using inovative heuristics. The architecture and the main concepts of MLSDO have been presented, then its algorithms and their goals have been described in detail. Afterwards, the setting and the sensitivity of its parameters, its computational complexity, the efficiency of its strategies, the dynamic adaptation of the number of its exploring and tracking agents, and its convergence have been studied through an extensive experimental analysis and discussion. Comparisons with competing algorithms on the well known Moving Peaks Benchmark, as well as on the Generalized Dynamic Benchmark Generator, show the efficiency of the proposed algorithm.

In works in progress, the MLSDO algorithm is applied to several real-world problems. Currently, we are working on its application to the segmentation, and to the registration of sequences of biomedical images. We plan to adapt MLSDO to dynamic combinatorial optimization. We would like also to make the critical parameters of MLSDO adaptive, *i.e.*, such that they will be automatically adjusted. Finally, as many real-world DOPs are multiobjective, or have many constraints that can be dynamic, the proposed algorithm may also be adapted to the dynamic multiobjective optimization.

We are pleased to share the source code of MLSDO and will make it publicly available at http://lissi.fr.

Appendix: Nomenclature of all the variables used

Name	Short description
α	number of evaluations that makes a time span in the benchmarks
A_i	archive of the last n_m initial positions of agents that are created or relocated, in MLSDO
A_m	archive of the local optima found by the agents, in MLSDO
Δ_i	size of the interval that defines the search space on the i^{th} axis in the "non-normalized" basis
δ_{ph}	parameter of MLSDO that defines the highest precision parameter of the stagnation criterion of the agents local searches
δ_{pl}	parameter of MLSDO that defines the lowest precision parameter of the stagnation criterion of the agents local searches
d	dimension of the search space
$\mathbf{D}$	direction vector of the preceding displacement of an agent, in MLSDO
$\mathbf{D}'$	direction vector of the current displacement of an agent, in MLSDO
$\mathbf{D}_n$	direction vector of the displacement of an agent at the n^{th} step of its local search, in MLSDO
$\mathbf{e}_i$	i^{th} unit vector of the "non-normalized" basis of the search space
$f(\mathbf{x})$	the objective function of a static optimization problem
$f(\mathbf{x},t)$	the objective function of a dynamic optimization problem
$f^*(t)$	value of the best solution found at the t^{th} evaluation since the last change in the objective function

Name	Short description
$f_i(t)$	value of the best solution found at the t^{th} evaluation of GDBG
$f_i^*(t)$	value of the global optimum at the t^{th} evaluation of GDBG
f_j^*	value of the global optimum for the j^{th} time span in the benchmarks
f_{ji}^*	value of the best solution found at the i^{th} evaluation of the j^{th} time span of MPB
F_k	k^{th} problem of GDBG
$\bar{f}_x^*$	average relative error of the best fitness found at the x^{th} evaluation of a time span of MPB
$fitness$	function that returns the value of the objective function of a given solution, in MLSDO
$g_k(\mathbf{x},t)$	the k^{th} inequality constraint of a dynamic optimization problem
$h_j(\mathbf{x},t)$	the j^{th} equality constraint of a dynamic optimization problem
$isNotUpToDate$	flag that indicates if a change in the objective function occurred since the detection of a given stored optimum
m	number of optima currently stored in the archive A_m
$mark_{max}$	maximal mark that can be obtained on the considered test case of GDBG
$mark_{pct}$	mark obtained on the considered test case of GDBG
max	function that returns the maximum value among several given values
min	function that returns the minimum value among several given values
N	number of agents currently existing during the execution of MLSDO
n_a	parameter of MLSDO that defines the maximum number of "exploring" agents
n_c	parameter of MLSDO that defines the maximum number of "tracking" agents created after the detection of a change
N_c	number of changes in the benchmarks
$N_e(j)$	evaluations performed on the j^{th} time span of MPB
n_m	capacity of the archives A_i and A_m
$\mathbf{O_c}$	a newly found optimum
oe	offline error used in MPB
op	overall performance used in GDBG
R	step size of an agent of MLSDO
r_e	parameter of MLSDO that defines the exclusion radius of the agents, and the initial step size of "exploring" agents
$r_i(t)$	relative error of the best fitness found at the t^{th} evaluation of the i^{th} run of GDBG
r_l	parameter of MLSDO that defines the initial step size of "tracking" agents
r_{new}	initial step size of an agent, in MLSDO (can be equal to either r_e or r_l)
$round$	function that rounds a given number to the nearest integer
s	change severity used in MPB
$\mathbf{S_c}$	current solution of an agent, in MLSDO
$\mathbf{S'_c}$	best candidate solution of an agent at the current step of its local search, in MLSDO
$\mathbf{S_{prev}}$	a candidate solution generated with $\mathbf{S_{next}}$ in the local search of an agent, in MLSDO
$\mathbf{S_{new}}$	the initial solution of the local search of an agent, in MLSDO
$\mathbf{S_{next}}$	a candidate solution generated with $\mathbf{S_{prev}}$ in the local search of an agent, in MLSDO
$\mathbf{S_w}$	worst candidate solution of an agent at the current step of its local search, in MLSDO
t	number of evaluations performed since the beginning of the tested algorithm
T_k	k^{th} change scenario of GDBG
u	number of equality constraints
U	value of the *cumulative dot product* used to adapt the step size of an agent, in MLSDO
$\mathbf{u}_i$	i^{th} vector of the "normalized" basis of the search space
U_n	value of the *cumulative dot product* of an agent at the n^{th} step of its local search, in MLSDO
v	number of inequality constraints
$\mathbf{x}$	a solution in the search space of an optimization problem
x_i	i^{th} coordinate of the solution vector $\mathbf{x}$

References

1. Bird S, Li X (2007) Using regression to improve local convergence. In: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE, Singapore, pp 592–599
2. Blackwell T, Branke J (2004) Multi-swarm optimization in dynamic environments. Lecture Notes in Computer Science 3005:489–500
3. Blackwell T, Branke J (2006) Multi-swarms, exclusion and anti-convergence in dynamic environments. IEEE Transactions on Evolutionary Computation 10(4):459–472
4. Branke J (1999) Memory enhanced evolutionary algorithms for changing optimization problems. In: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE, Washington, DC, USA, pp 1875–1882
5. Branke J (1999) The Moving Peaks Benchmark website. http://people.aifb.kit.edu/jbr/MovPeaks
6. Branke J, Kaußler T, Schmidt C, Schmeck H (2000) A multi-population approach to dynamic optimization problems. In: Proceedings of Adaptive Computing in Design and Manufacturing, Springer, Berlin, Germany, pp 299–308
7. Brest J, Zamuda A, Boskovic B, Maucec MS, Zumer V (2009) Dynamic optimization using self-adaptive differential evolution. In: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE, Trondheim, Norway, pp 415–422
8. Dréo J, Siarry P (2006) An ant colony algorithm aimed at dynamic continuous optimization. Applied Mathematics and Computation 181(1):457–467
9. Du W, Li B (2008) Multi-strategy ensemble particle swarm optimization for dynamic optimization. Information Sciences 178(15):3096–3109
10. de França FO, Zuben FJV (2009) A dynamic artificial immune algorithm applied to challenging benchmarking problems. In: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE, Trondheim, Norway, pp 423–430
11. Gardeux V, Chelouah R, Siarry P, Glover F (2009) Unidimensional search for solving continuous high-dimensional optimization problems. In: Proceedings of the IEEE International Conference on Intelligent Systems Design and Applications, IEEE, Pisa, Italy, pp 1096–1101
12. Gonzalez JR, Masegosa AD, Garcia IJ (2010) A cooperative strategy for solving dynamic optimization problems. Memetic Computing 3(1):3–14
13. Jin Y, Branke J (2005) Evolutionary optimization in uncertain environments – a survey. IEEE Transactions on Evolutionary Computation 9(3):303–317
14. Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks IV, IEEE, Perth, Australia, pp 1942–1948
15. Korosec P, Silc J (2009) The differential ant-stigmergy algorithm applied to dynamic optimization problems. In: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE, Trondheim, Norway, pp 407–414
16. Lepagnot J, Nakib A, Oulhadj H, Siarry P (2009) Performance analysis of MADO dynamic optimization algorithm. In: Proceedings of the IEEE International Conference on Intelligent Systems Design and Applications, IEEE, Pisa, Italy, pp 37–42
17. Lepagnot J, Nakib A, Oulhadj H, Siarry P (2010) A new multiagent algorithm for dynamic continuous optimization. International Journal of Applied Metaheuristic Computing 1(1):16–38
18. Li C, Yang S (2008) A generalized approach to construct benchmark problems for dynamic optimization. In: Proceedings of the 7th International Conference on Simulated Evolution and Learning, Springer, Melbourne, Australia, pp 391–400
19. Li C, Yang S (2009) A clustering particle swarm optimizer for dynamic optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE, Trondheim, Norway, pp 439–446
20. Li C, Yang S, Nguyen TT, Yu EL, Yao X, Jin Y, Beyer HG, Suganthan PN (2008) Benchmark generator for CEC 2009 competition on dynamic optimization. Tech. rep., University of Leicester, University of Birmingham, Nanyang Technological University
21. Li X (2004) Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference, Springer, Seattle, Washington, USA, pp 105–116
22. Li X, Branke J, Blackwell T (2006) Particle swarm with speciation and adaptation in a dynamic environment. In: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, Seattle, Washington, USA, pp 51–58

23. Liu L, Yang S, Wang D (2010) Particle swarm optimization with composite particles in dynamic environments. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 40(6):1634–1648
24. Lung RI, Dumitrescu D (2007) Collaborative evolutionary swarm optimization with a Gauss chaotic sequence generator. Innovations in Hybrid Intelligent Systems 44:207–214
25. Lung RI, Dumitrescu D (2008) ESCA: A new evolutionary-swarm cooperative algorithm. Studies in Computational Intelligence 129:105–114
26. Mendes R, Mohais A (2005) DynDE: A differential evolution for dynamic optimization problems. In: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE, Edinburgh, Scotland, pp 2808–2815
27. Moser I, Chiong R (2010) Dynamic function optimisation with hybridised extremal dynamics. Memetic Computing 2(2):137–148
28. Moser I, Hendtlass T (2007) A simple and efficient multi-component algorithm for solving dynamic function optimisation problems. In: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE, Singapore, pp 252–259
29. Novoa P, Pelta DA, Cruz C, del Amo IG (2009) Controlling particle trajectories in a multi-swarm approach for dynamic optimization problems. In: Proceedings of the International Work-conference on the Interplay between Natural and Artificial Computation, Springer, Santiago de Compostela, Spain, pp 285–294
30. Parrott D, Li X (2004) A particle swarm model for tracking multiple peaks in a dynamic environment using speciation. In: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE, San Diego, California, USA, pp 98–103
31. Parrott D, Li X (2006) Locating and tracking multiple dynamic optima by a particle swarm model using speciation. IEEE Transactions on Evolutionary Computation 10(4):440–458
32. Pelta D, Cruz C, Gonzalez JR (2009) A study on diversity and cooperation in a multiagent strategy for dynamic optimization problems. International Journal of Intelligent Systems 24(7):844–861
33. Pelta D, Cruz C, Verdegay JL (2009) Simple control rules in a cooperative system for dynamic optimisation problems. International Journal of General Systems 38(7):701–717
34. Tfaili W, Siarry P (2008) A new charged ant colony algorithm for continuous dynamic optimization. Applied Mathematics and Computation 197(2):604–613
35. Wang H, Wang D, Yang S (2009) A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems. Soft Computing 13(8-9):763–780
36. Yang S, Li C (2010) A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. IEEE Transactions on Evolutionary Computation 14(6):959–974
37. Yang S, Yao X (2008) Population-based incremental learning with associative memory for dynamic environments. IEEE Transactions on Evolutionary Computation 12(5):542–562
38. Yu EL, Suganthan P (2009) Evolutionary programming with ensemble of external memories for dynamic optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE, Trondheim, Norway, pp 431–438
39. Zeng S, Shi H, Kang L, Ding L (2007) Orthogonal Dynamic Hill Climbing Algorithm: ODHC. In: Yang S, Ong YS, Jin Y (eds) Evolutionary Computation in Dynamic and Uncertain Environments, Springer, pp 79–104