International Journal of Computational Intelligence Systems, Vol. 8, No. 6 (2015) 1053-1075

An adaptive local search with prioritized tracking for Dynamic Environments

A.D. Masegosa [1,2] *, E. Onieva[1] , P. Lopez-Garcia[1] , E. Osaba[1] , A. Perallos[1]

[1] *Deusto Institute of Technology, University of Deusto, 48007, Bilbao, Spain*

E-mail: {ad.masegosa,enrique.onieva,p.lopez,e.osaba,perallos}@deusto.es

[2] *IKERBASQUE, Basque Foundation for Science, 48011, Bilbao, Spain*

Received 20 May 2015

Accepted 30 August 2015

Abstract

Dynamic Optimization Problems (DOPs) have attracted a growing interest in recent years. This interest is mainly due to two reasons: their closeness to practical real conditions and their high complexity. The majority of the approaches proposed so far to solve DOPs are population-based methods, because it is usually believed that their higher diversity allows a better detection and tracking of changes. However, recent studies have shown that trajectory-based methods can also provide competitive results. This work is focused on this last type of algorithms. Concretely, it proposes a new adaptive local search for continuous DOPs that incorporates a memory archive. The main novelties of the proposal are two-fold: the prioritized tracking, a method to determine which solutions in the memory archive should be tracked first; and an adaptive mechanism to control the minimum step-length or precision of the search. The experimentation done over the Moving Peaks Problem (MPB) shows the benefits of the prioritized tracking and the adaptive precision mechanism. Furthermore, our proposal obtains competitive results with respect to state-of-the-art algorithms for the MPB, both in terms of performance and tracking ability.

Keywords: Dynamic Environments; Dynamic Optimization Problems; Trajectory-based Methods; Prioritized Tracking; Local Search; Adaptive Metaheuristics

1. Introduction

Many real-world optimization problems present some sort of dynamism in one or more of their components. Modelling them as static problems can result in an oversimplification of the reality that leads to low quality and little robust solutions. In this way, Dynamic Optimization Problems (DOPs) were proposed to deal with this dynamism and to provide models closer to real-world conditions. The excellent results of these approaches in many fields [1,36] have attracted a growing attention of the research community in the last decade. The interested reader is referred to [9, 29, 34] for recent reviews in DOPs.

When solving DOPs, three conditions are usually assumed: changes in the problem are gradual, they can be detected, and after a change, the problem remains static for a certain period. Due to the gradualness of the changes, the information acquired while solving the current static period of the problem can help to accelerate the search of the new optima after a change. Under these conditions, restarting the search from scratch after each change would waste information and worsen the performance.

*Corresponding author

Most of the approaches proposed for DOPs are population-based methods [4, 29] because their higher solution diversity has proof to provide good results in detecting and tracking changes. Among the population-based methods proposed for DOPs we can find Evolutionary Algorithms [10, 12, 28, 29, 38], Particle Swarm Optimization [4, 20, 21, 35], Ant Colony Optimization [23,24], Harmony Search [33], multi-agent metaheuristics [30, 31] or Portfolio Algorithms [8] to cite but a few. However, in recent years, there is a growing literature showing that trajectory-based methods, coupled with appropriate diversity mechanisms, can also provide very competitive results [14, 17, 22, 25].

In this work, we would like to deepen in this last type of approaches, to show that they not only provide competitive results, but also offer a good tracking ability. To this end, we propose a new adaptive local search for continuous DOPs. This method consists on a hill climbing that adapts the step length along the search and that stores the best solutions found in a memory archive. The search process of the proposed technique oscillates between two phases: tracking and exploration. The first one starts just after a change detection and aims at tracking the optima found in the last static period. To this end, the search is re-initialized from the solutions stored in the memory archive. The exploration phase begins when all solutions in the memory archive have been tracked. Its goal is to find new local optima by restarting the local search from random points.

Although there exist other works where trajectory-based methods with memory archives have been proposed [17, 25, 26, 37], our method presents two main novelties: the prioritized tracking and a mechanism to adjust the resolution or the minimum step-length of the search. The majority of the methods proposed so far in DOPs incorporate some sort of technique to facilitate the tracking of the optima after a change occurs (generating or maintaining diversity, keeping solutions in an external memory or dividing the population into several sub-populations). However, as far as we know, they do not incorporate mechanisms to establish what solutions or optima should be tracked first. This would allow providing good solutions for the new environment in a shorter time. This is the purpose of the prioritize tracking. In our proposal, before the tracking phase begins, the solutions stored in the memory archive are ranked in order to prioritize the exploration of the most promising areas in first place.

As mentioned before, the other novelty of our proposal is a mechanism to adjust the resolution of the search. This mechanism aims at balancing the maximum precision of the local search as function of the effort required to track the optima found. Concretely, it increases the precision (it reduces the minimum step-length) when the optima are easily tracked, in order to find solutions as close as possible to the exact position of the optima; and it decreases the precision otherwise, to avoid wasting time on approaching to the exact position of the optima found and foster the exploration of new ones.

The experimental study of this paper is done over the well-known Moving Peaks Benchmark (MPB) [6] and its goals are two-fold: 1) assessing the benefits of the prioritized tracking and the precision balance mechanisms, and 2) comparing the proposal with other state-of-the-art algorithms for the MPB.

The paper is structured as follows. The next section discusses the related work. Our proposal is presented in Section 3 where we show its general working as well as the prioritized tracking and precision balance mechanisms. Section 4 contains the experimental framework of this paper, describing the MPB problem, the implementation details, the performance measures and the non-parametric tests employed for statistical assessments. The analysis of the results and the comparison with the state-of-the-art algorithms are given in Section 5. Finally, Section 6 points out the main conclusions drawn from this work.

2. Related Work

In this section, we will review the literature related to our proposal, concretely those works where trajectory-based algorithms have been used as the main method to address DOPs.

One of the first in which a trajectory-based method was applied to DOPs can be found in [37].

Here, Zeng et al. proposed a method where several local searches explore and track the different optima of the problem. The best solutions found in each stage are stored in a memory archive to accelerate the searching process after a change. The working of the local searches is divided in two phases. In the first one, they explore new local optima whereas in the second one, they try to find solutions as close as possible to these optima.

Another interesting trajectory-based method for DOPs was presented in [27]. In this paper, Moser et al. use Extremal Optimization to search promising starting points for a Hill Climbing method. The solutions refined by the Hill Climbing are also stored in a memory to facilitate the tracking of the optima after changes. Moser together with Chiong presented an improved version of this algorithm in [26]. Basically, they combined the same initialization scheme with two others more sophisticated local searches, obtaining very good results for the MPB problem.

Multi-search methods (several trajectory-based methods combined in parallel) are another of the approaches successfully applied in DOPs. Lepagnot et al. proposed in [15] and [16] a multi-agent system where each agent implements a local search to explore the search space. A central coordinator controls the position of the agents and stores in an archive the best solutions found so far by them. These solutions are then used to relocate the agents when they get trapped in a local optimum. A new version of this algorithm was published by the same authors in [17], where the local searches implemented by the agents and some components of the cooperation scheme were improved.

A similar approach was presented in [14]. In this paper, Gonzalez et al. proposed a cooperative strategy composed by a set of agents that implement a Tabu Search, and a coordinator, that controls the location of the agents. The coordinator uses a rule base to decide when and where relocate the agents. The same authors use this method in others works to evaluate the performance of different control rules for the coordinator [13], and to study the influence of the number of agents and the neighbourhood sampling strategy [22].

Finally, a recent application of a trajectory-based method to DOPs can be found in [25]. The proposal consists on a local search algorithm called S3 that also keeps the best solutions found so far in a memory archive. When a change occurs, this method also uses the solutions stored in the archive, but it applies first a exponential crossover in order to help the search to escape from possible local optima.

Although the algorithm proposed in this paper shares some of the common components of the approaches mentioned above as an external memory archive, its design presents important novelties with respect to them. As pointed out before, the main difference are the mechanism to prioritize the tracking of the optima, and the method to adjust the resolution of the local search. Next section describes the proposal and these new mechanisms.

3. Description of adaptive local search

The method proposed in this paper consists on a local search for continuous DOPs with adaptive step size that incorporates specific mechanisms to deal with dynamism (e.g. fitness change detection), prioritized tracking of local optima and an adjustment method to balance the maximum precision of the search. For a better understanding, in Section 3.1 we give the general concepts and workflow of the method; in Sections 3.2 and 3.3, the details of the prioritized tracking and the precision adjustment mechanisms, respectively; and finally, in Section 3.4, the complete working of the algorithm.

3.1. Basic concepts and general workflow

Given that the method presented in this paper deals with continuous DOPs, an important issue to address in trajectory-based algorithms is the step length δ. In our proposal, the step length can also be seen as the neighbourhood size, since it establishes the area around the current solution that will be sampled in order to find better solutions. If δ is too small, the method will waste objective function evaluations in very short steps. On the contrary, if too long, it may have difficulties to converge to good solutions. For this reason, it is desirable a large step length at the beginning of the search, for a better exploration, and

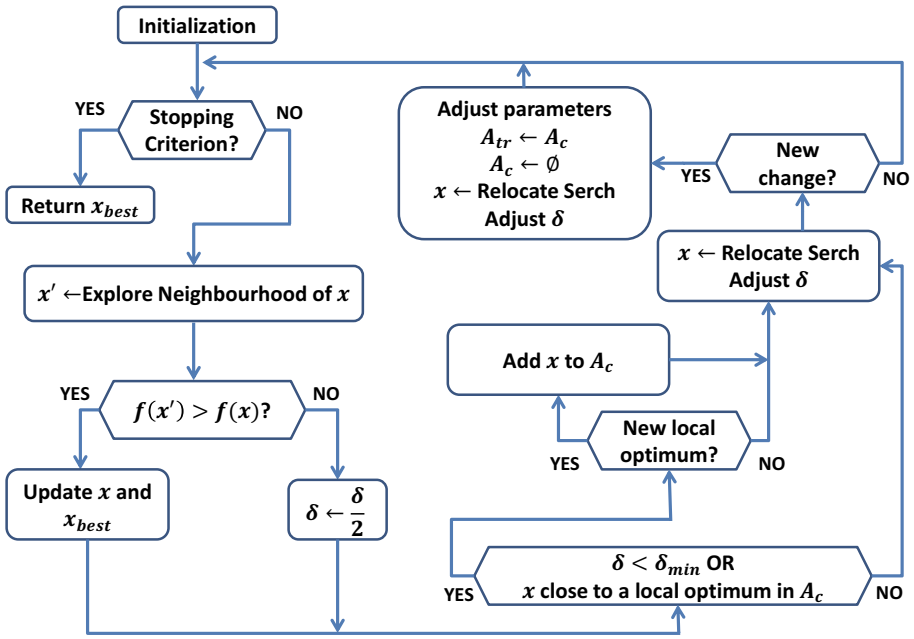


Fig. 1. Workflow of the adaptive local search

then, reducing it progressively to give more accurate steps. Having this idea in mind, the local search proposed starts with a step length $\delta = \delta_{init}$, and every time the exploration of the neighbourhood of the current solution does not lead to an improvement, δ is halved. The minimum step length is delimited by the parameter δ_{min}. This parameter determines the resolution or maximum precision of the search. As explained later in this section, δ_{min} is adjusted after each change in the fitness function.

Let us suppose that we are dealing with a maximization problem. When δ reaches a value lower than δ_{min}, we can assume that the search is placed in a local maximum, given that the method cannot find better solutions in a very small neighbourhood area. The local maxima found by the method along the search process are stored in a memory archive A_c, because of two reasons: 1) to avoid previously explored regions, and 2) to facilitate the tracking of the new positions of these local optima. When a change in the objective function is detected, the archive A_c is copied to another memory archive, called A_{tr}. A_c only stores the local maxima found in the current static period of the fitness function.

Algorithm 1: exploreNeighbourhood(x,δ,$maxNeighs$)

```
1  counter ← 0;
2  x_new ← x ;
3  repeat
4      x_neigh ← generate a random solution in the
       hypercube with side length 2δ and center in x;
5      if f(x_neigh) > f(x_new) then
6          x_new ← x_neigh ;
7          break;
8      end
9  until counter ⩾ maxNeighs;
10 counter ← counter + 1;
11 return x_new
```

Figure 1 displays the general workflow of the method. The algorithm starts with an initialization phase and after that, it enters into the main loop until the stopping condition is fulfilled. In the first step of this loop, the algorithm explores the neighbourhood of the current solution x. Algorithm 1 shows the pseudocode of the method employed by the local search to this end. As we can see, it generates points uniformly distributed in the area delimited by the hypercube with side length 2δ and centre in x. The exploration stops when it finds a solution x_{neigh} better than x or it samples a maximum number of

Algorithm 2: relocateSearch(A_{tr})

```
if A_tr ≠ ∅ then
    x_new ← first local optimum in A_tr;
    A_tr ← A_tr − {x_new};
else
    x_new ← generate random initial solution;
end
return x_new
```

neighbours (*maxNeighs*), returning x_{neigh} or x, respectively.

Coming back to Figure 1, after the neighbourhood exploration, the algorithm checks whether the returned solution x' is better than x. If so, it sets the current solution to x' and it updates the best solution found so far x_{best} in case $f(x') > f(x_{best})$; otherwise, δ is halved, since no improvement has been achieved.

In the next stage of the workflow, the method tests whether the step length δ is lower than δ_{min} or whether x is close to a local maximum stored in A_c. When any of these conditions is fulfilled, the algorithm checks if x is not close to a previously found local maximum in A_c (the euclidian distance is not lower than a prefixed value r_{lm}). This indicates that the method has found a new optimum, and x is added to the memory archive A_c.

After that, the algorithm restarts the search from another solution in order to find as many local optima as possible. Algorithm 2 describes how the new starting point of the search is selected. The rationale behind this relocation policy is simple. Right after a change, the method should start to explore regions near the solutions stored in A_{tr} (local maximum found in the last static period) since the new local optima will be probably close to them, given the gradualness of the changes. When all of these regions have been explored, restarting the search from random points will allow finding new local optima. More formally, if $A_{tr} \neq \emptyset$, x is set to the first solution in A_{tr}, which is then removed from A_{tr}; and otherwise, x is located randomly. It is important to note that the solutions in A_{tr} are sorted in order to prioritize the tracking of some optima over the others, as we will see later in Section 3.2.

Algorithm 3: readjustDelta()

```
if the method is in tracking phase then
    δ_new ← δ_track;
else
    δ_new ← δ_init;
end
return (δ_new)
```

The period in which the local search is restarted from the solutions in A_{tr} or from randomly generated solutions are called the tracking and the exploration phases, respectively. Depending on whether the method is in one phase or the other, the appropriate initial setting for the parameter δ varies. In the tracking phase, a small value for δ is preferred since we can expect the new local maxima to be close. However, in the second phase, it is better a bigger step length to increase the exploration and accelerate the convergence to new promising regions. Algorithm 3 displays the procedure used to readjust δ after the relocation of the search. δ_{track} and δ_{init} are the adjusted values for δ in the tracking and exploration phases, respectively.

Finally, in the last stage of the main loop, the local search checks whether a change in objective function has occurred or not. Concretely, the method revaluates a randomly picked local maximum from A_c if $A_c \neq \emptyset$, or it revaluates x_{best}, otherwise. When a change is detected, the method readapts certain parameters (we will explain this adaptations later), set A_{tr} to A_c, and resets A_c. After that, it relocates the search following the same procedure shown in Algorithm 2 and it updates x_{best}.

In the next two subsections, we will describe how the method prioritizes the tracking of the optima and it adapts the parameter δ_{min} in order to balance the precision of the search.

3.2. Prioritized tracking mechanism

3.2.1. Motivation

Apart from tracking as much local optima as possible, it is also important to track first the most promising local maxima to provide good solutions as soon as possible after a change. The question that arises now is what criterion follow to prioritize the exploration of the solutions stored in A_{tr} once a change is

detected. The intuitive approach would be to select each time the best solution according to the new fitness. However, the information from the past environment can also be useful since, due to the gradualness of the changes, a good optimum in the previous period has a high probability of being good in the current one.

To illustrate this fact, we performed some experiments using the Moving Peaks Benchmark with the standard configuration shown in Table 1 (see Section 4.1 for the formal definition of the benchmark and its standard configuration). Although this benchmark has not been defined yet, to understand this analysis, the reader only needs to know that the problem considered here consists on ten local maxima whose positions vary in each change; and that the parameter s, called *shift severity*, establishes how strong these changes are (the higher the value, the stronger the severity of the changes).

Concretely, the experimentation done with this problem aimed at simulating the ideal situation in which we find all local optima and we have their positions stored in the memory archive A_{tr}. Then, right after a change, in order to find the new global maximum as soon as possible, we have to decide which of the solutions in A_{tr} track first: the one with the highest fitness before or after the change.

To this end, we run the Moving Peaks Benchmark 30 times, with 100 changes per run, and considering six different severities of change ($s = 1,\ldots,6$). Let us assume that the positions of the 10 local optima at the stage t (static period between the $(t-1)$-th and the t-th changes) are $\{p_1(t),\ldots,p_{10}(t)\}$, and their fitness values are given by $\{f(p_1(t),t),\ldots,f(p_{10}(t),t)\}$. In each stage, we measured the portion of times that the i-th local optima is the global one in both the current and the previous stage, that is:

$$\exists i \; s.t. \; f(p_i(t-1),t-1) > f(p_k(t-1),t-1)$$
and
$$f(p_i(t),t) > f(p_k(t),t), \forall k \neq i, k = 1,\ldots,10$$

Furthermore, we also measured the portion of times in which the index of the position $p_i(t-1)$ with the highest fitness in the current stage t, also corresponds with the index of the best optimum in this stage, that is, the portion of times in which the next two conditions are fulfilled:

$$\exists i \; s.t. \; f(p_i(t-1),t) > f(p_k(t-1),t)$$
and
$$f(p_i(t),t) > f(p_k(t),t), \forall k \neq i, k = 1,\ldots,10$$

The portions measured correspond to the expected success rate we would have at predicting the starting point that would lead to track the global optima. The first measure corresponds with the expected success rate if the fitness in the previous stage is used as predictor, while the second one, with the expected success rate if the reevaluated fitness is considered.

Figure 2 displays the mean success rates, averaged over all changes and runs, for six different severities when the former (red-triangle series) and the current (black-circle series) fitness are considered as predictors, respectively. We can observe here that the success rate for the former fitness is the same regardless the severity of the changes. However, when the current fitness is considered, the accuracy of the predictions shows an important variation, obtaining good results for low severities but worsening as the severity increases. This fact indicates that when the severity is low, using the current information to choose the best solution to track is better. This happens because the landscape of the problem changes slightly. On the contrary, when the severity is high, the landscape changes sharply and the current fitness can make very noisy predictions. Therefore, in this case, the previous information is a better indicator.

Another possibility consists on ranking the solutions in A_{tr} by aggregating their current and former fitness. A simple way of doing it is given in Equation 1. The parameter α determines to what extent the previous fitness ($f(p_i(t-1),t-1)$) and the current one ($f(p_i(t-1),t)$) are consider to rank the solutions. A higher value of α gives a bigger importance to the former fitness, whereas a lower value gives it to the current one.

$$f_{agg}(p_i(t-1),t) = \alpha f(p_i(t-1),t-1) + (1-\alpha) f(p_i(t-1),t) \quad (1)$$

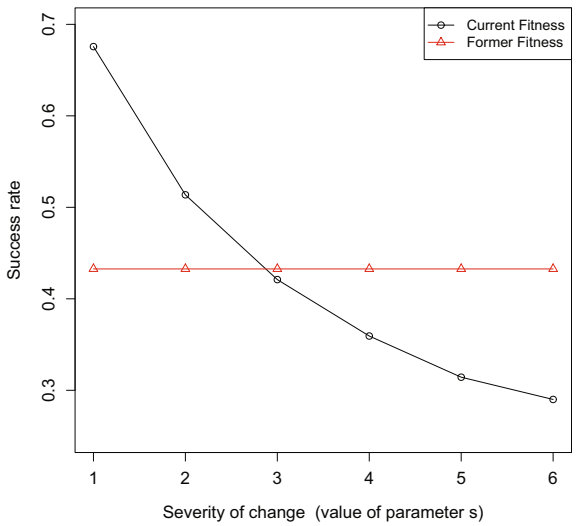


Fig. 2. Success rate of the prediction of the best global optima in the Moving Peaks Benchmark when the fitness of the local optima in the previous stage (former fitness series) and the fitness of their positions in the current stage (current fitness series) are considered

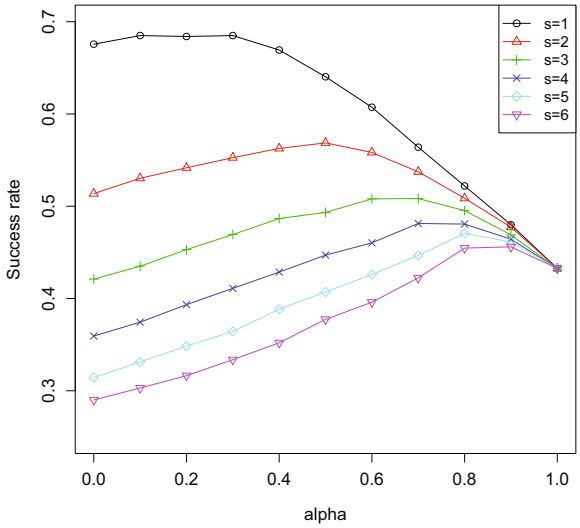


Fig. 3. Success rate, as a function of the parameter α, of the prediction of the best global optima in the Moving Peaks Benchmark using a linear combination described in Equation 1, for different severity values.

The success rates achieved with this aggregated measure for different severities of change are shown in Figure 3. We can observe that the combination of current and former information allows improving the success rates obtained using only the previous ($\alpha = 1$) or the current ($\alpha = 0$) fitness. This happens especially for intermediate severities ($s = 2,3,4$), when α is set properly. The best value for α grows almost proportionally to the severity, indicating that, as the severity increases, it is better to give more importance to the former fitness.

3.2.2. *Design of the prioritize tracking*

The expression shown in Equation 1 is the approach used by the local search to establish the best archived solution to track first. In order to adjust the parameter α to the features of the problem being solved, we employ the Equation 2. The functions max and min are used to delimit the value of α between 0 and 1. K_α is a parameter to set and $\widehat{s}$ is an estimation of the severity of the changes.

$$\alpha = \max(0, \min(K_\alpha \cdot \widehat{s}, 1)) \qquad (2)$$

This severity may be determined by different factors. One of the most common approaches to establish the severity of the changes in continuous DOPs is by the shift length of the position of the optima. For this reason, we designed the next procedure to calculate $\widehat{s}$. During the tracking phase, every time a solution in A_{tr} is selected as restarting point, such position is stored in the variable x_{init}. Then, when a local maximum is found, we assume that it is the new position of the optimum that was located in x_{init} at the former stage. Therefore, measuring the Euclidian distance between these two points, we have an estimation of the shift length for this optimum. The procedure to update $\widehat{s}$ along the search process is described in Algorithm 4. x corresponds to the position of the new local maximum found.

The procedure employed to sort A_{tr} using Equations 1 and 2 is displayed in Algorithm 5. The function $f_f(lm)$ returns the fitness of the solution lm before the change, that is, $f_f(lm) = f(lm, t-1)$.

Algorithm 4: updateSeverityEstimation(x_{init},x,$\widehat{s}$)

```
if the method is in tracking phase then
    d ← euclideanDistance(x_init, x);
    ŝ_new ← updateAvgSeverity(d, ŝ);
else
    ŝ_new ← ŝ;
end
return (ŝ_new)
```

Algorithm 5: sortTrackingArchive(A_{tr},$\widehat{s}$)

1 $\alpha = \max(0, \min(K_\alpha \cdot \widehat{s}, 1))$;
2 **for** $lm \in A_{tr}$ **do**
3 $\quad f_{agg}(lm) \leftarrow \alpha f_f(lm) + (1-\alpha) f(lm)$;
4 **end**
5 *sort* A_{tr} *according to the* $f_{agg}(lm)$ *values* ;
6 **return** (A_{tr})

3.3. Precision balance mechanism

As stated before, the parameter δ_{min} establishes the minimum step length of the local search. Once the algorithm has "fallen" in the basin of attraction of a local optimum, the value of this parameter will determine the precision with which the method will try to approach to exact position of the maximum. A low value for δ_{min} will allow to get closer to the local optimum, obtaining a better refinement of the solutions found, but at the expense of dedicating a greater number of function evaluations in each local optimum. On the contrary, a high value for δ_{min} will make the method spend less function evaluations in each local maximum (the parameter δ will reach the value δ_{min} in a lower number of cycles), at the expense of having worse approximations to the local maxima, and therefore, less quality solutions. When the number of local optima in the problem or the effort required to track them is low, the method can "afford" a better refinement, whereas in the opposite case, it is desirable to sacrifice this refinement in order to track a higher number of local optima or to find new ones.

Having this idea in mind, we have developed a mechanism to adapt the parameter δ_{min} on the basis of the effort required by the method to track all optima. This effort is measured by the portion of time required by the local search to track all the solutions stored in A_{tr}. Algorithm 6 shows the working of this mechanism that is run just after a change is detected. First, it calculates the ratio between time spent (measured in evaluations) in the tracking phase, e_{tp}, and the time (number of evaluations) since the last change, e_{lc}. If this ratio is bigger than a certain threshold μ_{tr}, it means that the algorithm is spending too much time in tracking local optima. In this case, the mechanism double the parameter δ_{min}

Algorithm 6: readjustResolution(e_{tp}, e_{lc}, δ_{min})

1 $r_{tr} \leftarrow e_{tp}/e_{lc}$
2 **if** $r_{tr} > \mu_{tr}$ **then**
3 $\quad \delta_{min}^{new} \leftarrow \min(\delta_{min} * 2, \delta_{min})$
4 **else**
5 $\quad \delta_{min}^{new} \leftarrow \max(\delta_{min}/2, \delta_{min})$
6 **end**
7 **return** (δ_{min}^{new})

to reduce the time dedicated to refine the search in each local maximum. When the ratio is lower than μ_{tr}, δ_{min} is halved to improve the refinement of the found solutions. The range of values for δ_{min} is limited to avoid too big and too small step lengths. In this way, δ_{min}^{lb} and δ_{min}^{ub} establish the lower and upper bounds for δ_{min}, respectively.

3.4. Overall working of the method

To conclude the description of the proposal, we show its overall working in Algorithm 7. The only aspect that we have not explained yet in this pseudocode is the motivation of the 'if condition' displayed in line 29. This conditional sentence aims at avoiding an excessive waste of function evaluations checking changes in the objective function. e_{curr} is the current number function evaluations performed by the local search, $e_{lastCheck}$ is the number of evaluations at which the last change check was done, and $E_{changeCheck}$ is a parameter that establishes the minimum number of evaluations required to allow another change detection.

4. Experimental Framework

This section is devoted to describe the experimental framework used in this work. Concretely, in Section 4.1 we will give the details of the problem used as benchmark, in Section 4.2 the performance measures, in Section 4.3 the non-parametric statistical tests used to assess the significance of the comparisons, and in Section 4.4 the implementation details and the parameter configuration of the adaptive local search.

Algorithm 7: Pseudo-code for the complete working of the adaptive local search

```
1  δ ← δ_init;
2  x ← generate random initial solution;
3  x_best ← x;
4  e_curr ← 0;
5  while not stopping condition do
6      x' ← exploreNeighborhood(x, δ, maxNeighbours);
7      if f(x') > f(x) then
8          x ← x' ;
9          if f(x) > f(x_best) then
10             x_best ← x ;
11         end
12     else
13         δ ← δ/2;
14     end
15     if δ < δ_min ∨ x is close to a local optimum in A_c then
16         if x is not close to a local optimum in A_c then
17             A_c ← A_c + {x} ;
18             ŝ ← updateSeverityEstimation(x_init, x, ŝ);
19         end
20         if A_tr = ∅ ∧ the method is in tracking phase then
21             update e_tp;
22             finish tracking phase;
23         end
24         x ← relocateSearch(A_tr);
25         δ ← readjustDelta();
26         x_init ← x;
27     end
28     update e_curr;
29     if (e_curr − e_lastCheck) > E_changeCheck then
30         e_lastCheck ← e_curr ;
31         if the fitness function has changed then
32             A_tr ← A_c;
33             A_c ← ∅;
34             δ_min ← readjustResolution(e_tp, e_lc, δ_min);
35             if A_tr ≠ ∅ then
36                 A_tr ← sortTrackingArchive(A_tr, ŝ);
37                 start tracking phase;
38             end
39             x ← relocateSearch(A_tr);
40             δ ← readjustDelta() ;
41             x_best ← x, x_init ← x ;
42         end
43     end
44 end
45 return x_best
```

4.1. Moving Peaks Benchmark

The Moving Peaks Benchmark (MPB) was originally proposed in [6], and it is one of the most known benchmarks for DOPs. It is a maximization problem consisting in the superposition of **m** peaks, each one characterized by its own height (**h**), width (**w**), and location of its center (**p**). The fitness function of the MPB is defined as follows:

$$\mathbf{f}(\mathbf{x},\mathbf{t}) = \max_i \left(h(t)_i - w(t)_i \sqrt{\sum_{j=1}^{n} (x^j - p_i^j(t))^2} \right) \quad (3)$$

where $i = 1,\ldots,m$ and **n** is the dimensionality of the problem. The highest point of each peak corresponds to its center. Therefore, the global optimum is the centre of the peak with the highest value for **h**. The dynamism is introduced by periodically changing all or some of the parameters of each peak *i* after every ω function evaluations:

$$\mathbf{h}_i(t+1) = \mathbf{h}_i(t) + \mathbf{h_s} \cdot \mathbf{N}(0,1) \quad (4)$$

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \mathbf{w_s} \cdot \mathbf{N}(0,1) \quad (5)$$

$$\mathbf{p}_i(t+1) = \mathbf{p}_i(t) + \mathbf{v}_i(t+1) \quad (6)$$

$$\mathbf{v}_i(t+1) = \frac{\mathbf{s}}{|\mathbf{r} + \mathbf{v}_i(t)|}((1-\lambda)\mathbf{r} + \lambda \mathbf{v}_i(t)) \quad (7)$$

where $\mathbf{h}_i(t)$, $\mathbf{w}_i(t)$ and $\mathbf{p}_i(t)$ are the height, width and position of the $i-th$ peak at the time t. Changes to both peaks width and height depend on the parameters $\mathbf{w_s}$ and $\mathbf{h_s}$, respectively, which control the severity of the changes. The variation of the peak position depends on a shift vector $\mathbf{v}_i(t+1)$, which is a linear combination of a random vector **r** and the previous shift vector $\mathbf{v}_i(t)$ for the peak, normalized to the value of the parameter **s** called shift severity. The parameter **s** determines the length of the peak shifts in each change. The random vector **r** is created by drawing uniformly distributed random numbers for each dimension and normalizing its length to **s**. Finally, the parameter λ establishes the linear correlation with respect to the previous shift. A value of 1 indicates "total correlation" and a value of 0 "pure randomness".

Table 1. Parameter configuration for the MPB problem

Parameter	Value
number of peaks (**m**)	10
change frequency (ω)	5000 evaluations
height severity ($\mathbf{h_s}$)	7.0
width severity ($\mathbf{w_s}$)	1.0
peak shape	cone
basic function	no
shift length (**s**)	1.0
number of dimensions (**n**)	5
correlation coefficient (λ)	0
range for variables' domain	[0, 100]
range for $\mathbf{h}_j(t)$	[30.0, 70.0]
range for $\mathbf{w}_j(t)$	[1, 12]
initial values for $\mathbf{h}_j(t)$	50.0

To facilitate the comparison with other peer methods, we set MPB parameters to standard values which are shown in Table 1. Unless otherwise stated, the parameters will remain the same in all the experimentation done.

4.2. Performance Measures

In order to make our results comparable with other closely related works, we have selected the next performance measures:

- *Offline error* (e_{off}) [7]: this measure is the average of the error of the best solution found by the algorithm since the last change, for every function evaluation, and for all changes. Given that the changes in the environment are produced at a fixed rate, its formula is:

$$e_{off} = \frac{1}{N_r N_c N_e} \sum_{i=1}^{N_r} \sum_{j=1}^{N_c} \sum_{k=1}^{N_e} (f_{ij}^* - f_{ijk}) \quad (8)$$

  where N_r in the number of runs, N_c is the total number of changes in the environment, $N_e = \omega$ is the number of function evaluations of the change frequency, f_{ij}^* is the fitness value of the optimum solution in the *i*-th run and *j*-th change, and f_{ijk}^{best} is the fitness of the best solution found by the method since the beginning of the *j*-th change of the *i*-th run, up to the *k*-th fitness evaluation.
  In this work, each run uses a different random seed, and we have chosen $N_r = 50$ and $N_c = 100$.

- *Best error before change* (e_{bbc}) [7]: this measure is the average of the best error at the fitness evaluation just before a change occurs. Following the former notation, this measure is defined as:

$$e_{bbc} = \frac{1}{N_r N_c} \sum_{i=1}^{N_r} \sum_{j=1}^{N_c} (f_i^* - f_{ijN_e}) \tag{9}$$

- *tRatio*: this measure aims at quantifying the tracking ability of the algorithm. Corresponds to the average ratio of peaks tracked by the method over all runs and changes:

$$tRatio = \frac{total\ different\ peaks\ tracked}{\mathbf{m} \cdot N_r \cdot N_c} \tag{10}$$

  A peak is considered to be tracked if the algorithm finds a solution at a distance lower than 0.1 from the peak.
- *gRatio*: this measure aims at determining how well algorithm tracks the global optima. It is defined as the portion of times the best local maximum is tracked (following the same criterion as above) over all runs and changes:

$$gRatio = \frac{total\ times\ best\ peak\ is\ tracked}{N_r \cdot N_c} \tag{11}$$

4.3. Non-parametric tests

Given that we are dealing with stochastic methods, we need to use statistical tests to assess if the differences in performance that appear among the studied algorithms are significant or not. In this article we follow the guidelines proposed in [11], where non-parametric statistical testing is suggested in situations like the one faced in this contribution (several instances/problem configurations, algorithms and parameter settings).

In this work, we have employed two types of non-parametric tests, matched and unmatched. The unmatched non-parametric tests were used to check whether exist or not significant differences between two or more methods in one specific MPB configuration. The matched non-parametric tests were applied to make the same assessment over a set of problem configurations.

Concretely, as unmatched and matched non-parametric tests, in this order, we have applied Kruskal-Wallis and Friedman's tests for multiple comparisons among a set of methods; and Mann-Whitneys's and Wilcoxon's tests for pairwise comparisons. Regarding many-to-many and one-to-many comparisons, we have used the Holm's post-hoc test procedure for the adjustment of the p-values. To carry out the unmatched and matched tests, we employed the R package Agricolae[†] and the software tool KEEL [2], respectively.

Table 2. Parameter setting for the adaptive local search

Parameter	Value
max. number of neighbours (*maxNeighs*)	10
initial value for δ (δ_{init})	20
value for δ in tracking phase (δ_{track})	0.5
constant to adjust α (K_α)	0.15
threshold for tracking ratio (μ_{tr})	0.7
upper bound for δ_{min} (δ_{min}^{ub})	$n \cdot 10^{-1}$
lower bound for δ_{min} (δ_{min}^{lb})	$2n \cdot 10^{-4}$
dist. for closeness to local opt. in A_c (r_{lm})	10
min. evals. for checking changes ($E_{changeCheck}$)	10

4.4. Implementation Details

Table 2 displays the parameter settings of the proposed adaptive local search for the experimentation. These parameters have been set according to previous studies and experiments.

For the MPB benchmark, we employed the original implementation provided by J. Branke[‡]. The method has been implemented in Java 8 and its source code can be downloaded at the link available in the footnote[§]

5. Experimental Results

The experimentation done in this work has the following objectives:

[†] `http://cran.r-project.org/web/packages/agricolae/index.html` (Accessed September 2015)
[‡] `https://code.google.com/p/moving-peaks/source/browse/trunk/MPB/`(Accessed September 2015)
[§] `http://research.mobility.deustotech.eu/media/publication_resources/alsptde_source.zip`

1. Analyse the benefits of the prioritized tracking.
2. Study the benefits of the precision adjustment mechanism.
3. Compare the performance and tracking ability of the proposal with state-of-the-art methods for the MPB.

The remaining part of this section is devoted to analyse the results of this experimentation.

5.1. *Benefits of the prioritized tracking*

In this subsection we will study the effects of the prioritize tracking in the proposed method. To this end, we considered the next four variants of the adaptive local search:

- **NPT**: no prioritized tracking is employed. The solutions in A_{tr} are tracked in a FIFO order, according to when they were found (first found, first tracked).
- **PTC**: prioritized tracking using the current fitness to order the solutions in A_{tr}.
- **PTF**: prioritized tracking using the former fitness to order the solutions in A_{tr}.
- **PTC+F**: prioritize tracking using the Equations 1 and 2 to order the solutions in A_{tr}.

Aiming at measuring the benefits of prioritized tracking without any distortion, in this experiments, the mechanism to adjust the parameter δ_{min} was not activated. We tested the four variants over six MPB configurations with different shift severity values ($s = 1,\ldots,6$). Table 3 shows the mean ranks returned by the Kruskal-Wallis test for the global performance of the method over the six configurations of the problem, for each of the four measures outlined before. For e_{off} and e_{bbc}, the lower the mean rank the better, whereas for *tRatio* and *gRatio* the higher the better. This test established that the null hypothesis can be rejected at a confidence level of 0.95 for the e_{off} measure only. The superscripts of the values indicate whether there exist differences among the methods at a significant level of 0.05, applying Holm's p-value corrections for many-to-many comparisons. If two values in the same row present a different superscript, their difference in performance is significant respect to that measure. Otherwise, the null hypothesis of similar performance cannot be rejected.

Table 3. Mean ranks returned by the Kruskal-Wallis non-parametric test for the global performance of NPT, PTC, PTF and PTC+F over the MPB with six different shift severities ($s = 1,\ldots,6$). The superscripts indicate if there are differences among the methods at a significant level of 0.05 (Holm's procedure). Different and equal superscripts in the same row indicate significant and non-significant differences for that measure, respectively. Best mean rankings are highlighted in bold.

Measure	NPT	PTC	PTF	PTC+F
e_{off}	1014.9^{a}	541.3^{b}	462.3^{c}	$\mathbf{383.5^{d}}$
e_{bbc}	609.9^{a}	$\mathbf{588.7^{a}}$	611.0^{a}	592.5^{a}
tRatio	596.0^{a}	$\mathbf{603.5^{a}}$	602.8^{a}	599.7^{a}
gRatio	591.6^{a}	603.8^{a}	602.2^{a}	$\mathbf{604.5^{a}}$

The data of the table show that the prioritized tracking only has significant effects in terms of offline error. These results indicates that, globally, the prioritized tracking improve the efficiency of the method (e_{off}) but not the efficacy (e_{bbc}) or the tracking ability (*tRatio* and *gRatio*). In other words, for these MPB configurations, the proposed mechanism allows tracking faster the best optima but not finding better solutions before changes nor more local maxima. In any case, the efficiency represents one of the most important aspects in this type of problems, since one of the main goal of DOPs methods consists on providing good solutions as soon as possible after a change. For this reason, the offline error and offline performance are the measures most widely used in DOPs.

Focusing on the (e_{off}) mean ranks, the three types of prioritized tracking significantly outperforms the variant of the proposal without this mechanism. Among the three types of prioritized tracking, PTC+F is the best method, which shows that the combination of current and past information provides more accurate predictions of the expected quality of the local optima in the new environment.

Table 4 displays the e_{off}, e_{bbc}, *tRatio* and *gRatio* obtained by these four algorithms in each of the six MPB configurations considered. The Kruskal-Wallis test for multiple comparisons determines that

Table 4. e_{off}, e_{bbc}, *tRatio* and *gRatio* obtained by NPT, PTC, PTF and PTC+F over six MPB configurations with different shift severities ($s = 1, \ldots, 6$). The superscripts of the values provide the same information explained in Table 3.

s		NPT	PTC	PTF	PTC+F
1	e_{off}	1.27^{a}	0.44^{c}	0.56^{b}	$\mathbf{0.43^{c}}$
	e_{bbc}	0.14^{a}	0.14^{a}	0.14^{a}	$\mathbf{0.13^{a}}$
	tRatio	$\mathbf{0.97^{a}}$	$\mathbf{0.97^{a}}$	$\mathbf{0.97^{a}}$	$\mathbf{0.97^{a}}$
	gRatio	0.97^{a}	0.97^{a}	$\mathbf{0.98^{a}}$	$\mathbf{0.98^{a}}$
2	e_{off}	1.85^{a}	$0.54^{b,c}$	0.66^{b}	$\mathbf{0.50^{c}}$
	e_{bbc}	0.15^{a}	0.10^{a}	0.14^{a}	$\mathbf{0.09^{a}}$
	tRatio	0.96^{a}	$\mathbf{0.97^{a}}$	$\mathbf{0.97^{a}}$	$\mathbf{0.97^{a}}$
	gRatio	0.97^{a}	$\mathbf{0.98^{a}}$	0.97^{a}	$\mathbf{0.98^{a}}$
3	e_{off}	2.22^{a}	0.78^{b}	$0.77^{b,c}$	$\mathbf{0.66^{c}}$
	e_{bbc}	0.14^{a}	$\mathbf{0.12^{a}}$	0.17^{a}	0.13^{a}
	tRatio	$\mathbf{0.96^{a}}$	$\mathbf{0.96^{a}}$	$\mathbf{0.96^{a}}$	$\mathbf{0.96^{a}}$
	gRatio	$\mathbf{0.97^{a}}$	$\mathbf{0.97^{a}}$	$\mathbf{0.97^{a}}$	$\mathbf{0.97^{a}}$
4	e_{off}	2.58^{a}	1.07^{b}	0.81^{c}	$\mathbf{0.76^{c}}$
	e_{bbc}	$\mathbf{0.15^{a}}$	0.18^{a}	$\mathbf{0.15^{a}}$	$\mathbf{0.15^{a}}$
	tRatio	$\mathbf{0.95^{a}}$	$\mathbf{0.95^{a}}$	$\mathbf{0.95^{a}}$	$\mathbf{0.95^{a}}$
	gRatio	$\mathbf{0.97^{a}}$	$\mathbf{0.97^{a}}$	$\mathbf{0.97^{a}}$	0.96^{a}
5	e_{off}	2.75^{a}	1.32^{b}	0.92^{c}	$\mathbf{0.87^{c}}$
	e_{bbc}	0.20^{a}	0.17^{a}	$\mathbf{0.16^{a}}$	0.18^{a}
	tRatio	$\mathbf{0.93^{a}}$	$\mathbf{0.93^{a}}$	$\mathbf{0.93^{a}}$	$\mathbf{0.93^{a}}$
	gRatio	$\mathbf{0.96^{a}}$	$\mathbf{0.96^{a}}$	$\mathbf{0.96^{a}}$	$\mathbf{0.96^{a}}$
6	e_{off}	3.01^{a}	1.63^{b}	1.08^{c}	$\mathbf{1.02^{c}}$
	e_{bbc}	$\mathbf{0.23^{a}}$	$\mathbf{0.23^{a}}$	$\mathbf{0.23^{a}}$	$\mathbf{0.23^{a}}$
	tRatio	0.90^{a}	$\mathbf{0.91^{a}}$	$\mathbf{0.91^{a}}$	$\mathbf{0.91^{a}}$
	gRatio	0.94^{a}	$\mathbf{0.95^{a}}$	$\mathbf{0.95^{a}}$	0.94^{a}

there are significant differences (p-value < 0.05) among the algorithms only in terms of offline error, for all severity values. The superscripts provide the same information as above but restricted to the corresponding severity configuration of the problem. The first issue to highlight is the strong improvement achieved by the prioritized tracking in terms of e_{off}, regardless the shift severity, and being significant in all cases. Comparing PTC and PTF, as expected, the first one obtains better results for low severity values ($s = 1,2$; being significant for $s = 1$) while worse for higher values ($s = 3, \ldots, 6$; being significant for $s = 4,5,6$). The mechanism that combines both fitness, PTF+C, leads to the lowest e_{off} in the six MPB configurations. The null hypothesis of similar performance can be rejected for PTC when the shift severity is higher or equal to four, whereas for PTF, when it is lower or equal to two. Regarding the remaining performance measures, as we said before, the difference among the methods are not significant

Table 5. Mean ranks returned by the Kruskal-Wallis test for the global performance of FR_{lb}, FR_{ub} and AR over the MPB problem with eight different number of peaks ($peaks = \{5,10,20,30,40,50,100,200\}$). The superscripts of the values provide the same information explained in Table 3.

Measure	FR_{lb}	FR_{ub}	AR
e_{off}	689.4^{a}	618.1^{b}	$\mathbf{494.0^{c}}$
e_{bbc}	673.5^{a}	633.2^{a}	$\mathbf{494.8^{b}}$
tRatio	449.8^{b}	$\mathbf{677.0^{a}}$	674.7^{a}
gRatio	529.2^{b}	615.7^{a}	$\mathbf{656.6^{a}}$

in any case. Therefore, the results confirm the benefits of the prioritized tracking in terms of e_{off}, and the better accuracy of PTF+C, versus PTC and PTF, in determining the local maxima that should be tracked first.

5.2. Benefits of precision adjustment mechanism

This second subsection studies the benefits of the mechanism developed to adjust the precision or resolution parameter δ_{min} (described in Section 3.3). As stated above, it controls the minimum step length in order to balance the maximum precision of the search according to the effort required to track the local optima. Concretely, it increases δ_{min} when the effort required is low, to favour the refinement of solutions, and it decreases it when high, to favour the exploration of new optima. The experimentation was designed in order to assess if the mechanism meets its target or not. More specifically, we compared three versions of the adaptive local search. Two, (FR_{lb}) and (FR_{ub}), where the resolution parameter δ_{min} was fixed to its lower and upper bounds, δ_{min}^{lb} and δ_{min}^{ub}, along the whole search process, respectively. In the other one (AR), δ_{min} is adjusted according to the mentioned procedure. The three variants were tested over eight configurations of the MPB with 5, 10, 20, 30, 40, 50, 100 and 200 peaks.

Table 5 displays the mean ranks provided by the Kruskal-Wallis test for the global performance, over the eight MPB configurations, of the three variants of the adaptive local search considered in this experimentation, with respect to the measures e_{off}, e_{bbc}, *tRatio* and *gRatio*. This test reports that at least two methods present significant different in performance (p-value < 0.05) for all measures. The reader should

take into account again that for e_{off} and e_{bbc}, the lower the mean rank the better, whereas for *tRatio* and *gRatio*, the higher the better. The best mean ranking in each case is highlighted in bold. The superscripts have the same meaning as before.

The mean ranks shows clearly the benefits of the adjustment resolution mechanism in all performance aspects considered. *AR* significantly improves FR_{lb} and FR_{ub} in terms of e_{off} and e_{bbc} which indicates that the mechanism enhances the efficacy and efficiency of the method. Regarding the tracking ability, *AR* performs significantly better than FR_{lb}, the variant with a higher refinement balance, and very similar to FR_{ub}, the variant with a higher exploration balance. Therefore, the proposed adjustment method allows improving the efficacy and efficiency of the local search, keeping a good tracking ability.

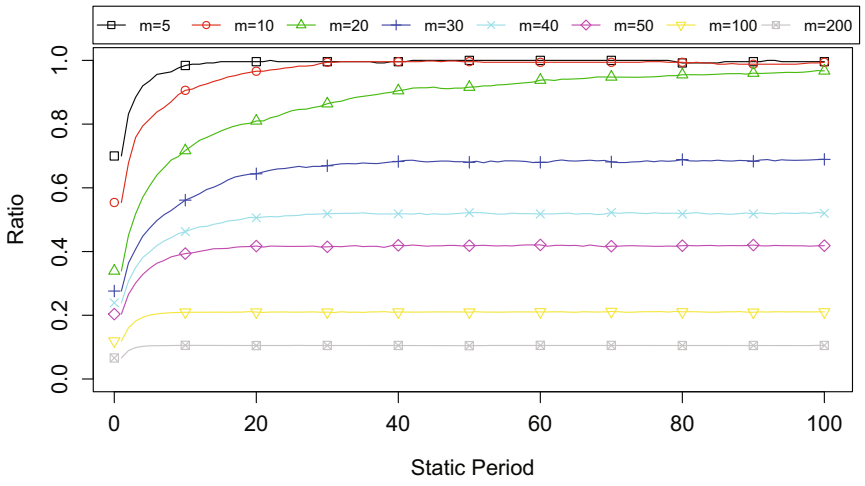
We will now continue analysing the results obtained in each MPB configuration separately. Table 6 contains the e_{off}, e_{bbc}, *tRatio* and *gRatio* obtained by the three methods in each of the eight MPB configurations studied. The superscripts keep showing the statistical significance of the difference among the methods. Let us compare first FR_{ub} and FR_{lb}. Looking at the e_{off} and e_{bbc} values, FR_{lb} shows a better performance when the number of peaks is low ($m = 5, 10, 20$; being significant in all cases) but worse when the number of peaks is high ($m = 30, 40, 50, 100, 200$; being significant for $m \geqslant 40$). Regarding the tracking measures, *tRatio* and *gRatio*, the results are rather similar, excepting that the null hypothesis can and cannot be rejected for $m = 20$ and $m = 30$, respectively. These data confirm what pointed out above: when the effort required to track the optima is low (small number of peaks) a higher precision is preferred (the method can "afford" a better refinement of the solutions). In the opposite scenario (high number of peaks), a lower precision helps to find and track more local optima. Focusing on the results of *AR*, this method achieves a good precision balance in all cases. It obtains virtually the same performance than FR_{lb} and FR_{ub} in configurations with low ($m \leqslant 20$) and high ($m \geqslant 40$) number of peaks, respectively; and it improves them significantly when the number of peaks is intermediate ($m = 30$).

Table 6. e_{off}, e_{bbc}, *tRatio* and *gRatio* obtained by FR_{lb}, FR_{ub} and *AR* over 8 MPB configuration with different number of peaks ($peaks = \{5, 10, 20, 30, 40, 50, 100, 200\}$) when the mechanism to adjust the resolution parameter δ_{min} is activated and not. The superscripts of the values provide the same information as explained in Table 3.

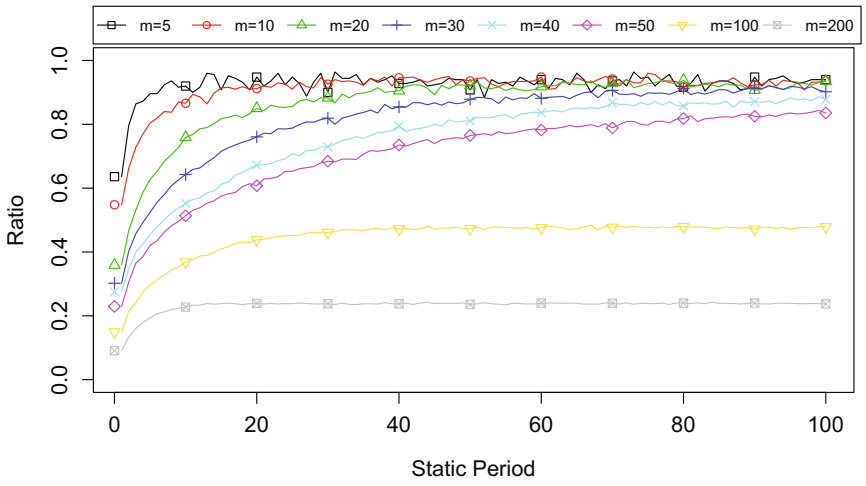
peaks(m)	Measure	FR_{lb}	FR_{ub}	*AR*
5	e_{off}	$\mathbf{0.41^b}$	0.72^a	$\mathbf{0.41^b}$
	e_{bbc}	$\mathbf{0.10^b}$	0.50^a	$\mathbf{0.10^b}$
	tRatio	$\mathbf{0.99^a}$	0.93^b	$\mathbf{0.99^a}$
	gRatio	$\mathbf{0.99^a}$	0.92^b	$\mathbf{0.99^a}$
10	e_{off}	$\mathbf{0.41^b}$	0.73^a	$\mathbf{0.41^b}$
	e_{bbc}	$\mathbf{0.10^b}$	0.50^a	$\mathbf{0.10^b}$
	tRatio	$\mathbf{0.97^a}$	0.91^b	$\mathbf{0.97^a}$
	gRatio	$\mathbf{0.97^a}$	0.92^b	$\mathbf{0.97^a}$
20	e_{off}	0.56^b	0.76^a	$\mathbf{0.53^b}$
	e_{bbc}	0.30^b	0.50^a	$\mathbf{0.20^b}$
	tRatio	0.87^b	0.87^b	$\mathbf{0.89^a}$
	gRatio	$0.90^{a,b}$	0.88^b	$\mathbf{0.91^a}$
30	e_{off}	0.91^a	0.80^a	$\mathbf{0.69^b}$
	e_{bbc}	0.60^a	0.60^a	$\mathbf{0.40^b}$
	tRatio	0.65^b	$\mathbf{0.82^a}$	0.81^a
	gRatio	0.68^b	$\mathbf{0.84^a}$	0.83^a
40	e_{off}	1.06^a	0.81^b	$\mathbf{0.79^b}$
	e_{bbc}	0.80^a	0.60^b	$\mathbf{0.50^b}$
	tRatio	0.50^c	$\mathbf{0.76^a}$	0.74^b
	gRatio	0.54^b	$\mathbf{0.79^a}$	0.77^a
50	e_{off}	1.18^a	$\mathbf{0.82^b}$	0.84^b
	e_{bbc}	0.90^a	0.60^b	$\mathbf{0.60^b}$
	tRatio	0.41^c	$\mathbf{0.71^a}$	0.68^b
	gRatio	0.47^b	$\mathbf{0.75^a}$	0.73^a
100	e_{off}	1.45^a	$\mathbf{0.88^b}$	0.92^b
	e_{bbc}	1.20^a	$\mathbf{0.70^b}$	$\mathbf{0.70^b}$
	tRatio	0.21^c	$\mathbf{0.45^a}$	0.43^b
	gRatio	0.25^b	$\mathbf{0.50^a}$	0.48^a
200	e_{off}	1.57^a	$\mathbf{0.95^b}$	0.99^b
	e_{bbc}	1.30^a	$\mathbf{0.80^b}$	$\mathbf{0.80^b}$
	tRatio	0.10^b	$\mathbf{0.23^a}$	0.23^a
	gRatio	0.13^b	$\mathbf{0.27^a}$	0.26^a

In the next part of this subsection, we will get more insight into the reasons behind the good performance of the resolution adjustment mechanism. Figures 4 A, B and C show the evolution of the mean ratio between the number of peaks tracked in each static period and the total number of peaks in the corresponding configuration, for FR_{lb}, FR_{ub} and *AR*, respectively. The data are averaged over the 50 runs. To establish when a peak was tracked, we used the same criterion than for *tRatio*. The static period and

A) FR_{lb}



B) FR_{ub}



C) AR

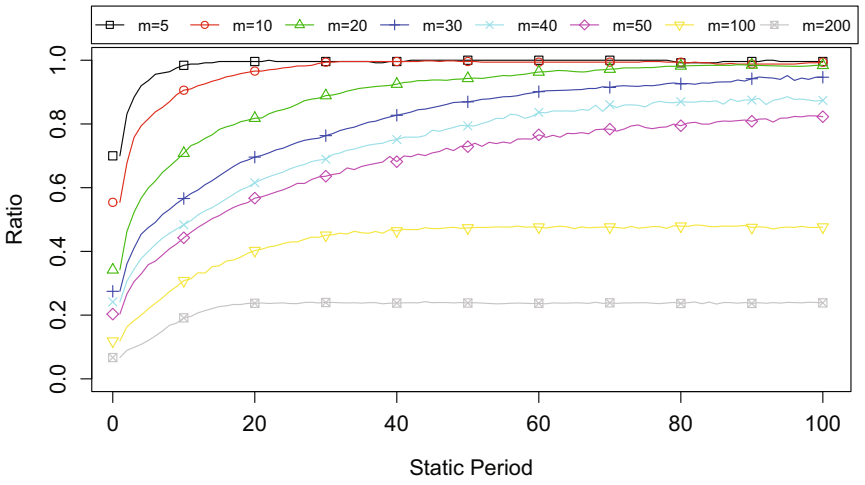


Fig. 4. Evolution of the mean ratio between the number of peaks tracked in each static period and the total number of peaks in the corresponding MPB configuration ($\frac{\text{peaks tracked}}{m}$) for FR_{lb} (A), FR_{ub} (B) and AR (C). The static period and the mean ratio are represented in the x and y axis, respectively. The eight series corresponds to the eight MPB configurations with different number of peaks (m).

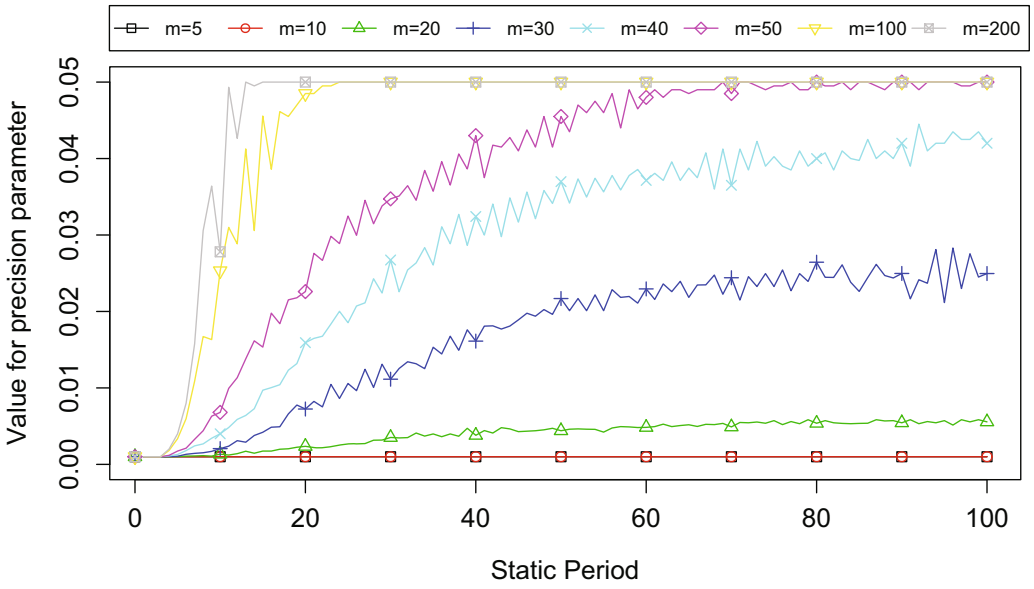


Fig. 5. Evolution of the adjustment of the parameter δ_{min} along the search process depending on the number of peaks (m) of the MPB configuration. The x-axis represents the static period of the objective function and the y-axis depicts the value for this parameter. The series indicate the corresponding MPB configuration.

the mean ratio are represented in the x and y axis, in this order. The series corresponds to the eight MPB configurations with different number of peaks m.

Let us focus again on FR_{lb} and FR_{ub}. For a low number of peaks ($m = 5, 10, 20$), we can observe that, although the evolution of the mean ratio at the first stationary periods is similar, as search progresses, FR_{lb} is able to track nearly all optima. However, FR_{ub} stagnates at around a 90% of the optima. Its low precision does not allow it to reach all local maxima at a distance lower than 0.1 (distance to consider a peak as tracked). For a high number of peaks ($m \geqslant 30$), the situation is the opposite. The mean ratio for FR_{lb} gets stagnated in the first static periods of the search at a much lower value than the one obtained by FR_{ub}. When FR_{lb} finds a certain number of peaks (around 20 peaks according to the mean ratio and the value of m), it is not able to track more. This happens because it spends all the static period trying to get solutions closer to the exact position of these optima.

On the contrary, the plot for AR shows that this method tracks as many optima as the best of the other two variants, in each of the MPB configurations. For 5, 10 and 20 peaks, its behaviour is virtually the same than that of FR_{lb} whereas, in the remaining configurations, the same goes for FR_{ub}. Comparing AR with FR_{ub}, we can also see that the growth of the curve is slightly slower for AR. This indicates that AR needs some time to adjust the parameter δ_{min} to the appropriate value.

This fact is illustrated in Figure 5, that displays how the proposed mechanism adjusts the value of δ_{min} along the search process. The x-axis represents the static period of the objective function and the y-axis depicts the value for this parameter. The series indicate the corresponding MPB configuration. The plot shows that for $m = 5$ and $m = 10$, the mechanism sets δ_{min} to the minimum value (δ_{min}^{lb}), to increase the precision of the search. For $m = 20$, $m = 30$ and $m = 40$, it progressively adjusts the resolution to intermediate values between δ_{min}^{lb} and δ_{min}^{ub}. Finally, for $m = 50$, $m = 100$ and $m = 200$, δ_{min} is continually increased (with different velocities) up to it reaches δ_{min}^{ub}. In short, this graphic illustrates how the proposed mechanism adjusts the maximum

Table 7. Acronym, bibliographic reference and description of the state-of-the-art algorithms considered in the comparison

Acronym	Reference	Description
CPSO	[35]	Clustering particle swarm optimizer for locating and tracking multiple optima
mCPSO	[5]	Multiswarm with charged particles as well as exclusion and anti-convergence mechanisms
mQSO	[5]	Multiswarms with quantum particles as well as exclusion and anti-convergence mechanisms
SOS+LS	[3]	Self-Organizing scouts algorithm coupled with local search
DynPopDE	[12]	Differential Evolution for dynamic environments with unknown numbers of optima
ESCA	[21]	Multi-population hybrid between a Particle Swarm Optimization method and a Evolutionary Algorithm
CPSOR	[19]	General framework of multipopulation methods with clustering for undetectable dynamic environments
CHPSO	[32]	Hybrid adaptive collaborative approach based on particle swarm optimization and local search
AMSO	[20]	Adaptive multi-swarm optimizer for dynamic optimization problems
MLSDO	[17]	Multiple local search algorithm for continuous dynamic optimization problems
DynS3	[25]	Adaptive local search with memory archive for continuous dynamic optimization problems

precision of the local search depending on the effort required to track the local optima.

5.3. Comparison with the state-of-the-art algorithms

Once we have seen the benefits of the prioritized tracking and the mechanism to balance the precision of the search, in this section we aim at comparing the competitiveness of our proposal versus state-of-the-art algorithms for the MPB, both in terms of performance and tracking ability.

Table 7 shows the acronyms, references and descriptions of the state-of-the-art algorithms selected for this comparison. These algorithms have been chosen because they are known to be high performance methods for the MPB and they have been tested using the same standard configuration for this benchmark (see Table 1). The first nine algorithms are population-based. Concretely, CPSO, mCPSO, mQSO, CPSOR and AMSO are multi-swarm algorithms, that is, Particle Swarm Optimization (PSO) methods whose population (swarm) is divided into various sub-populations; SOS+LS is a Self-Organizing Scouts algorithm coupled with a local search; DynPopDE is a multi-population differential evolution; ESCA is a multi-population hybrid between a PSO method and a Evolutionary Algorithm; and CHPSO is a PSO algorithm combined with a local search. Finally, we also considered two recently proposed trajectory-based methods: MLSDO, a cooperative strategy with multiple local searches, and DynS3, an adaptive local search with memory archive. The interested reader is referred to the citations provided in Table 7 for further details about these state-of-the-art algorithms.

To compare the performance of our proposal against these methods, we have used the measure e_{off} because it is an standard in DOPs. We have not consider the e_{bbc} because it was not available for MLSDO and DynS3. The results for the first eight population-based algorithms (CPSO, mCPSO, mQSO, SOS+LS, DynPopDE, ESCA, CPSOR and AMSO) were taken from [20]. The authors of this work assert that they re-implemented and reproduced the original results of these methods over the same MPB configurations used here. The results for CHPSO, MLSDO and DynSE were taken from their original works ([32], [17] and [25], respectively).

We are aware that comparing the algorithms using the results reported in their respective papers can bias the comparison, because the sequence of changes in the objective function may be different. To minimize this problem, we have carefully chosen those algorithms and results where the experimentation was done according to the same standard configurations of the MPB and the same experimentation settings used in this article (50 runs per algorithm and MPB configuration). This comparison methodology has been recently used in other works as [17, 25, 32, 33]. Furthermore, some preliminary experiments with our method showed that, under the same MPB configuration, when different batched of 50 runs were compared among them, the variations in performance were very low.

Table 8. e_{off} and standard deviation obtained by our proposal (ALSPT) and the considered state-of-the-art algorithm in the MPB with different shift severity values ($s = 1 \ldots 6$).

Algorithm	$s=1$	$s=2$	$s=3$	$s=4$	$s=6$	$s=6$
ALSPT	0.43 ± 0.19	$\mathbf{0.50 \pm 0.13}$	$\mathbf{0.66 \pm 0.24}$	$\mathbf{0.77 \pm 0.22}$	$\mathbf{0.83 \pm 0.19}$	$\mathbf{1.09 \pm 0.26}$
CPSO	4.5 ± 0.26	5.4 ± 0.24	6.1 ± 0.19	6.7 ± 0.21	7.2 ± 0.15	7.6 ± 0.2
mCPSO	8.6 ± 1.1	9.1 ± 0.91	10 ± 0.69	12 ± 0.63	13 ± 0.77	14 ± 0.61
mQSO	2.8 ± 0.19	3.3 ± 0.14	3.8 ± 0.14	4.4 ± 0.14	5 ± 0.14	5.5 ± 0.13
DynPopDE	2.3 ± 0.79	2.5 ± 0.49	2.9 ± 0.62	3.8 ± 0.61	4.4 ± 0.54	4.7 ± 0.58
ESCA	15 ± 1.8	13 ± 1.1	13 ± 0.94	13 ± 0.92	13 ± 0.78	13 ± 0.78
CPSOR	2.6 ± 0.2	3.7 ± 0.19	4.5 ± 0.24	5.5 ± 0.26	6.1 ± 0.21	6.6 ± 0.25
AMSO	1.4 ± 0.11	2.2 ± 0.13	2.9 ± 0.18	3.4 ± 0.19	3.8 ± 0.16	4.2 ± 0.15
CHPSO	0.64 ± 0.02	0.92 ± 0.01	1.22 ± 0.02	1.50 ± 0.02	1.82 ± 0.01	-
MLSDO	$\mathbf{0.36 \pm 0.08}$	0.60 ± 0.07	0.92 ± 0.10	1.22 ± 0.12	1.62 ± 0.13	2.01 ± 0.19
DynS3	2.32 ± 0.01	2.84 ± 0.01	3.45 ± 0.03	4.00 ± 0.04	4.62 ± 0.04	5.33 ± 0.04

Table 9. e_{off} and standard deviation obtained by our proposal (ALSPT) and the considered state-of-the-art algorithm in the MPB with different number of peaks ($m = \{5, 10, 20, 30, 50, 100, 200\}$)

Algorithm	$m=5$	$m=10$	$m=20$	$m=30$	$m=50$	$m=100$	$m=200$
ALSPT	$\mathbf{0.40 \pm 0.15}$	0.43 ± 0.19	$\mathbf{0.55 \pm 0.13}$	$\mathbf{0.71 \pm 0.17}$	$\mathbf{0.84 \pm 0.14}$	$\mathbf{0.92 \pm 0.11}$	$\mathbf{0.98 \pm 0.10}$
CPSO	4.2 ± 0.32	4.5 ± 0.26	4 ± 0.16	3.5 ± 0.16	3.5 ± 0.13	3.2 ± 0.13	2.5 ± 0.091
mCPSO	7.3 ± 1.2	8.6 ± 1.1	8.6 ± 0.96	6.4 ± 0.72	6.4 ± 0.74	6.4 ± 0.46	6.0 ± 0.85
mQSO	2.6 ± 0.24	2.8 ± 0.19	3.4 ± 0.24	3.8 ± 0.46	3.7 ± 0.19	4.2 ± 0.33	4.3 ± 0.39
DynPopDE	2.0 ± 0.68	2.3 ± 0.79	2.3 ± 0.27	1.9 ± 0.28	2.1 ± 0.24	2.2 ± 0.33	2.0 ± 0.21
ESCA	13 ± 1.8	15 ± 1.8	11 ± 1.5	9.9 ± 1.1	10 ± 1.6	11 ± 1.3	8.5 ± 0.7
CPSOR	2.9 ± 0.34	2.6 ± 0.2	2.6 ± 0.3	2 ± 0.14	2.4 ± 0.12	2.5 ± 0.1	2.3 ± 0.097
AMSO	1.6 ± 0.28	1.4 ± 0.11	2 ± 0.19	1.5 ± 0.1	2 ± 0.16	2.1 ± 0.16	1.9 ± 0.17
CHPSO	0.44 ± 0.02	0.64 ± 0.02	0.91 ± 0.01	0.99 ± 0.01	1.03 ± 0.01	1.04 ± 0.01	1.01 ± 0.00
MLSDO	–	$\mathbf{0.36 \pm 0.08}$	–	–	–	–	–
DynS3	–	2.32 ± 0.01	2.65 ± 0.03	3.03 ± 0.03	3.27 ± 0.02	1.98 ± 0.01	2.11 ± 0.01

Table 10. e_{off} and standard deviation obtained by our proposal (ALSPT) and the considered state-of-the-art algorithms in the MPB with different dimensions ($n = \{5, 10, 20, 30, 50, 100\}$

Algorithm	$n=5$	$n=10$	$n=20$	$n=30$	$n=50$	$n=100$
ALSPT	0.43 ± 0.19	$\mathbf{1.63 \pm 0.92}$	$\mathbf{3.16 \pm 1.43}$	$\mathbf{4.03 \pm 1.79}$	$\mathbf{5.52 \pm 1.77}$	$\mathbf{12.79 \pm 2.81}$
CPSO	4.5 ± 0.26	8.9 ± 0.53	17 ± 1.4	25 ± 2.4	89 ± 5.1	–
mCPSO	8.6 ± 1.1	21 ± 2.3	70 ± 15	$1.9 \cdot 10^2 \pm 17$	$2.7 \cdot 10^2 \pm 20$	–
mQSO	2.8 ± 0.19	7.4 ± 0.36	14 ± 0.52	17 ± 0.31	33 ± 0.92	–
DynPopDE	2.3 ± 0.79	8.4 ± 1.4	13 ± 2.1	13 ± 2.4	18 ± 1.4	–
ESCA	15 ± 1.8	17 ± 4.1	47 ± 6.9	72 ± 20	$1.2 \cdot 10^2 \pm 10$	–
CPSOR	2.6 ± 0.2	9.6 ± 1	37 ± 5.6	70 ± 15	$1.4 \cdot 10^2 \pm 4.4$	–
AMSO	1.4 ± 0.11	4.2 ± 0.5	6.5 ± 1.4	8.1 ± 1.3	25 ± 3.7	–
CHPSO	0.64 ± 0.02	3.32 ± 0.06	5.04 ± 0.07	–	9.95 ± 0.11	–
MLSDO	$\mathbf{0.36 \pm 0.08}$	–	–	–	–	14.00 ± 2.33
DynS3	2.32 ± 0.01	–	–	–	–	28.04 ± 0.10

Table 11. Average rankings provided by the Friedman's non-parametric test for ALSPT, CPSOR, AMSO and CHPSO considering the results over 17 MPB configurations ($s = \{1,2,3,4,5\}$; $m = \{5,10,20,30,50,100,200\}$; $n = \{5,10,20,30,50\}$), and adjusted p-value according to the Holm's post-hoc procedure using ALSPT as control algorithm.

Algorithm	Avg. Ranking	p-value adj. Holm
ALSPT	1	-
CPSOR	4	0.0
AMSO	3	0.00008
CHPSO	2	0.04

Tables 8, 9 and 10 show the e_{off} measure and its standard deviation obtained by the ten state-of-the-art algorithms and our proposal in different configurations of the MPB regarding shift severity, number of peaks and dimension, in that order. Our proposal has been identified by the acronym ALSPT (Adaptive Local Search with Prioritized Tracking). The symbol '-' indicates that the corresponding e_{off} has not been reported. The best result in each MPB configuration is highlighted in bold.

Let us starting comparing the methods for different shift severity values. Table 8 shows that ALSPT obtains the best offline error in all cases, except for $s = 1$ where it is improved by MLSDO, the second best performing method over these configurations. These results also prove that ALSPT presents a better tolerance to the increase of the shift severity, since its performance experiences a much lower deterioration than the other algorithms. In our opinion, this is due to the good adaptation of the prioritized tracking to the shift severity of the changes.

Looking at Table 9, we can observe that our proposal also improves the state-of-the-art algorithms for all numbers of peaks but one (MLSDO in the same configuration as above). In this case, CHPSO is the method with the closest performance to ALSPT, especially when the number of peaks is high.

The results displayed in Table 10 show that our proposal also provides a very good scalability in terms of dimension. It always obtains the best offline error when the dimension is equal or greater to ten. Given that we only have available the average performance of the state-of-the-art algorithms, following the guidelines given in [11], we have used matched non-parametric test to assess the significance of the differences in performance. Since the results in all MPB configurations were not available for all methods, and the ratio between samples (e_{off} values available) and algorithms is low, we have splited the analysis in two parts. On one hand, we have compared ALSPT with the three best performing population-based algorithms, CHPSO, AMSO and CPSOR; and on the other hand, with the two trajectory-based algorithms, MLSDO and DynS3.

Table 12. Average rankings provided by the Friedman's non-parametric test for ALSPT, MLSDO and DynS3 considering the results over 7 MPB configurations ($s = \{1,2,3,4,5,6\}$; $n = \{100\}$), and adjusted p-value according to the Holm's post-hoc procedure using ALSPT as control algorithm.

Algorithm	Avg. Ranking	p-value adj. Holm
ALSPT	1.14	-
MLSDO	1.85	0.18
DynS3	3	0.001

Table 11 contains the results of the non-parametric tests. The second column depicts the average ranking provided by the Friedman's non-parametric test for ALSPT and the three best performing population-based methods. The third column displays the adjusted p-values according to the Holm's post-hoc test using ALSPT as control algorithm. This analysis have been done over the 17 MPB configurations where the results were available for all of them ($s = \{1,2,3,4,5\}$; $m = \{5,10,20,30,50,100,200\}$ and $n = \{5,10,20,30,50\}$). Friedman's test reported that the null hypothesis of similar performance could be rejected at significance level lower than 0.05. The average rankings show that, globally, ALSPT obtains better results than the three population based methods. Furthermore, the Holm's post-hoc procedure establishes that the difference in performance of ALSPT with respect to the three algorithms are significant (p-value < 0.05).

Table 12 shows the same information for the comparison among ALSPT, MLSDO and DynS3, over the seven MPB configurations where the results for three algorithms were available ($s = \{1,2,3,4,5,6\}$ and $n = \{100\}$). The Friedman's non-parametric test also determines that the null hypothesis can be rejected at a confidence level higher

Table 13. *tRatio* and standard deviation obtained by ALSPT and four population-based state-of-the-art algorithms considered in the MPB benchmark for different number of peaks. The standard deviations for AMSO are not displayed because they were not available in the literature.

Algorithm	$m = 10$	$m = 20$	$m = 30$	$m = 50$	$m = 100$	$m = 200$
ALSPT	$\mathbf{0.97 \pm 0.03}$	$\mathbf{0.89 \pm 0.05}$	$\mathbf{0.81 \pm 0.05}$	$\mathbf{0.68 \pm 0.03}$	$\mathbf{0.43 \pm 0.01}$	$\mathbf{0.23 \pm 0.001}$
CPSO	0.61 ± 0.1	0.41 ± 0.05	0.3 ± 0.04	0.2 ± 0.02	0.1 ± 0.01	0.053 ± 0.008
mQSO	0.18 ± 0.1	0.099 ± 0.05	0.086 ± 0.04	0.06 ± 0.03	0.03 ± 0.01	0.016 ± 0.007
CPSOR	0.73 ± 0.1	0.48 ± 0.1	0.42 ± 0.09	0.3 ± 0.06	0.16 ± 0.04	0.086 ± 0.02
AMSO	$0.92\pm$	$0.68\pm$	$0.63\pm$	$0.41\pm$	$0.22\pm$	$0.12\pm$

Table 14. *gRatio* and standard deviation obtained by ALSPT and four population-based state-of-the-art algorithms in the MPB benchmark for different number of peaks.

Algorithm	$m = 10$	$m = 20$	$m = 30$	$m = 50$	$m = 100$	$m = 200$
ALSPT	$\mathbf{0.97 \pm 0.003}$	$\mathbf{0.91 \pm 0.002}$	$\mathbf{0.83 \pm 0.002}$	$\mathbf{0.73 \pm 0.001}$	$\mathbf{0.48 \pm 0.001}$	$\mathbf{0.26 \pm 0.000}$
CPSO	0.71 ± 0.3	0.61 ± 0.2	0.48 ± 0.3	0.3 ± 0.2	0.2 ± 0.2	0.13 ± 0.2
mQSO	0.17 ± 0.3	0.053 ± 0.1	0.097 ± 0.2	0.076 ± 0.1	0.031 ± 0.06	0.025 ± 0.06
DynPopDE	0.63 ± 0.4	0.47 ± 0.4	0.38 ± 0.4	0.25 ± 0.3	0.12 ± 0.2	0.064 ± 0.1
AMSO	0.96 ± 0.09	0.74 ± 0.2	0.73 ± 0.3	0.45 ± 0.3	0.21 ± 0.2	0.13 ± 0.2

than 0.95. In this case, ALSPT also obtains the best average ranking, although the Holm's post-hot procedure reports that it is only significantly better than DynS3. The reader should note that there are only seven MPB configurations to compare three methods. Under such conditions (low ratio between the number of algorithms and the number of datasets), these tests require very strong differences in performances to reject the null hypothesis.

To finish this section, we will compare ALSPT with population-based state-of-the-art algorithms in terms of tracking ability. For this analysis, we could only consider those algorithms whose *tRatio* and *gRatio* results were available. Regarding the *tRatio* measure, its value was provided for CPSO, mQSO, CPSO and AMSO. For the first three algorithms, we took the results from the reference [18], whereas for the fourth one, from the reference [20] (in this case the standard deviations were not given in the paper). As for the *gRatio* measure, the data were available for CPSO, mQSO, DynPopDE and AMSO and all were extracted from [18]. As stated before, we carefully checked that the MPB configurations and the experimental settings were exactly the same.

The means and standard deviations of the *tRatio* and *gRatio* obtained by our proposal and the state-of-the-art algorithms mentioned above, in MPB configurations with different number of peaks, are presented in Tables 13 and 14, respectively. The best result in each case is highlighted in bold. Both tables show that ALSPT tracks more local optima and finds the global maximum more times than the population-based methods, especially in those MPB configurations with a high number of peaks. In this case, we cannot apply non-parametric test to assess the significance of the results, given that the number of MPB configurations is very low with respect to the number of methods compared. In any case, the strong differences between ALSPT and the other methods suggest that the differences are significant. Our aim with this comparison is to show that a trajectory-based algorithm can also provide a very good tracking ability when it is endowed with the appropriate mechanisms.

6. Conclusions

In this work we aimed at deepening in the use of trajectory-based algorithms in DOPs. To this end, we have proposed a local search technique with memory archive, whose main novelties were two adaptive mechanisms: one to prioritize the tracking of the solutions stored in the memory archive to accelerate the finding of the best optima after the changes; and another one, to balance the precision or resolution of the search depending on the effort required to track the optima.

The method was tested over the well-known

MPB benchmark and compared versus state-of-the-art algorithms for DOPs. From the analysis of the results we have drawn the next conclusions:

- The prioritized tracking improved significantly the efficiency of the method, that is, it allowed finding the best local optima in a lower time. The enhancement was achieved considering the following three types of prioritization: by the fitness in the former static period, by the fitness in the current one and by a linear combination of both of them.
- When comparing the prioritization modes based on the current and the former fitness, the first one provided better results in MPB configurations with low shift severity, whereas the second one, in those with high shift severity.
- The mode based on a linear combination of the current and former fitness offered more robust predictions about the best solutions to track, which led to a significantly better offline error than the other two modes when the global performance was considered.
- The mechanism to balance the maximum precision of the local search improved the performance of the method in all measures considered (e_{off}, e_{bbc}, *tRatio* and *gRatio*), especially when the number of local optima in the configuration of the MPB is high.
- The trajectory-based algorithm proposed obtained a lower offline error than the state-of-the-art algorithms considered in nearly all MPB configurations used in the experimentation.
- Our proposal showed a more competitive tracking ability than population-based state-of-the-art algorithms for the MPB problem.

In our opinion, the results obtained in this work show that trajectory-based algorithms deserve more attention in DOPs since, when provided with the proper adaptive mechanisms, they can lead to very competitive results with respect to population-based methods, even in terms of tracking ability.

Another interesting point of the research presented is that the two adaptive mechanisms proposed can be also applied to multi-population algorithms. For example, the prioritized tracking can be used to determine the order in which the sub-populations are evolved. Those sub-populations tracking the most promising optima can be evolved first to provide best solutions faster. On the other hand, some multi-population methods use diversity measures to determine when to stop a subpopulation because it has converged. The threshold that determines this convergence plays a similar role than the parameter δ_{min} in our proposal. In this way, our mechanism to adjust the precision of the search could also be used to adjust this threshold.

Acknowledgments

This work has been supported by the research projects TEC2013-45585-C2-2-R and TIN2014-56042-JIN from the Spanish Ministry of Economy and Competitiveness, and PC2013-71A from the Basque Government.

References

1. E. Alba, A. Nakib, and P. Siarry, editors. *Metaheuristics for Dynamic Optimization*, volume 433 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, 2013.
2. J. Alcalá-Fdez, L. Sánchez, S. García, M. J. del Jesus, S. Ventura, J. M. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas, J. C. Fernández, and F. Herrera. KEEL: a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3):307–318, 2008.
3. D. Ayvaz, H. R. Topcuoglu, and F. Gurgen. Performance evaluation of evolutionary heuristics in dynamic environments. *Applied Intelligence*, 37(1):130–144, 2011.
4. T. Blackwell. Particle Swarm Optimization in Dynamic Environments. In S. Yang, Y.-S. Ong, and Y. Jin, editors, *Evolutionary Computation in Dynamic and Uncertain Environments*, volume 51 of *Studies in Computational Intelligence*, pages 29–49. Springer Berlin Heidelberg, 2007.
5. T. Blackwell and J. Branke. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10(4):459–472, 2006.
6. J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Proceedings of the 1999 IEEE Congress on Evolutionary Computation (CEC-1999)*, pages 1875–1882, 1999.

7. J. Branke and H. Schmeck. Designing evolutionary algorithms for dynamic optimization problems. In *Advances in evolutionary computing*, Natural Computing Series, pages 239–262. 2003.
8. J. F. Calderín, A. D. Masegosa, and D. A. Pelta. Algorithm portfolio based scheme for dynamic optimization problems. *International Journal of Computational Intelligence Systems*, 8(4):667–689, 2015.
9. C. Cruz, J. González, and D. Pelta. Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing*, 15(17):1–22, 2010.
10. S. Das, A. Mandal, and R. Mukherjee. An adaptive differential evolution algorithm for global optimization in dynamic environments. *IEEE Transactions on Cybernetics*, 44(6):966–78, 2014.
11. J. Derrac, S. García, D. Molina, and F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.
12. M. C. du Plessis and A. P. Engelbrecht. Differential evolution for dynamic environments with unknown numbers of optima. *Journal of Global Optimization*, 55(1):73–99, 2012.
13. J. R. González, A. D. Masegosa, I. G. del Amo, and D. Pelta. Cooperation rules in a trajectory-based centralised cooperative strategy for dynamic optimisation problems. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC-2010)*, pages 1–8, 2010.
14. J. R. González, A. D. Masegosa, and I. J. García. A cooperative strategy for solving dynamic optimization problems. *Memetic Computing*, 3(1):3–14, 2011.
15. J. Lepagnot, A. Nakib, H. Oulhadj, and P. Siarry. Performance analysis of mado dynamic optimization algorithm. In *Proceedings of the 9th International Conference on Intelligent Systems Design and Applications (ISDA2009)*, pages 37–42, 2009.
16. J. Lepagnot, A. Nakib, H. Oulhadj, and P. Siarry. A new multiagent algorithm for dynamic continuous optimization. *International Journal of Applied Metaheuristic Computing*, 1(1):16–38, 2010.
17. J. Lepagnot, A. Nakib, H. Oulhadj, and P. Siarry. A multiple local search algorithm for continuous dynamic optimization. *Journal of Heuristics*, 19(1):35–76, 2013.
18. C. Li, T. T. Nguyen, M. Yang, S. Yang, and S. Zeng. Multi-population methods in unconstrained continuous dynamic environments: The challenges. *Information Sciences*, 296:95–118, 2015.
19. C. Li and S. Yang. A general framework of multipopulation methods with clustering in undetectable dynamic environments. *IEEE Transactions on Evolutionary Computation*, 16(4):556–577, 2012.
20. C. Li, S. Yang, and M. Yang. An adaptive multiswarm optimizer for dynamic optimization problems. *Evolutionary computation*, 22(4):559–94, 2014.
21. R. I. Lung and D. Dumitrescu. Evolutionary swarm cooperative optimization in dynamic environments. *Natural Computing*, 9(1):83–94, 2009.
22. A. D. Masegosa, D. Pelta, and I. G. del Amo. The role of cardinality and neighborhood sampling strategy in agent-based cooperative strategies for dynamic optimization problems. *Applied Soft Computing*, 14, Part C:577–593, 2014.
23. M. Mavrovouniotis and S. Yang. A memetic ant colony optimization algorithm for the dynamic travelling salesman problem. *Soft Computing*, 15(7):1405–1425, 2010.
24. M. Mavrovouniotis and S. Yang. Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors. *Applied Soft Computing*, 13(10):4023–4037, 2013.
25. M. Mavrovuniotis, F. Neri, and S. Yang. An Adaptive Local Search Algorithm for Real-Valued Dynamic Optimization. In *Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC-2015)*, 2015. In press.
26. I. Moser and R. Chiong. Dynamic function optimisation with hybridised extremal dynamics. *Memetic Computing*, 2(2):137–148, 2009.
27. I. Moser and T. Hendtlass. A simple and efficient multi-component algorithm for solving dynamic function optimisation problems. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC-2007)*, pages 252–259, 2007.
28. R. Mukherjee, G. R. Patra, R. Kundu, and S. Das. Cluster-based differential evolution with Crowding Archive for niching in dynamic environments. *Information Sciences*, 267:58–82, 2014.
29. T. T. Nguyen, S. Yang, and J. Branke. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6:1–24, 2012.
30. D. Pelta, C. Cruz, and J. R. González. A study on diversity and cooperation in a multiagent strategy for dynamic optimization problems. *International Journal of Intelligent Systems*, 24(7):844–861, 2009.
31. D. Pelta, C. Cruz, and J. L. Verdegay. Simple control rules in a cooperative system for dynamic optimisation problems. *International Journal of General Systems*, 38(7):701–717, 2009.
32. A. Sharifi, J. K. Kordestani, M. Mahdaviani, and M. R. Meybodi. A novel hybrid adaptive collaborative approach based on particle swarm optimization and local search for dynamic optimization problems. *Applied Soft Computing*, 32:432–448, 2015.
33. A. M. Turky and S. Abdullah. A multi-population harmony search algorithm with external archive for dynamic optimization problems. *Information Sciences*,

272:84–95, 2014.

34. S. Yang, Y. Jiang, and T. T. Nguyen. Metaheuristics for dynamic combinatorial optimization problems. *IMA Journal of Management Mathematics*, 24(4):451–480, 2012.
35. S. Yang and C. Li. A Clustering Particle Swarm Optimizer for Locating and Tracking Multiple Optima in Dynamic Environments. *IEEE Transactions on Evolutionary Computation*, 14(6):959–974, 2010.
36. S. Yang and X. Yao, editors. *Evolutionary Computation for Dynamic Optimization Problems*, volume 490 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, 2013.
37. S. Zeng, H. Shi, L. Kang, and L. Ding. Orthogonal dynamic hill climbing algorithm: ODHC. In S. Yang, Y.-S. Ong, and Y. Jin, editors, *Evolutionary Computation in Dynamic and Uncertain Environments*, volume 51 of *Studies in Computational Intelligence*, pages 79–104. Springer Berlin Heidelberg, 2007.
38. X. Zuo and L. Xiao. A DE and PSO based hybrid algorithm for dynamic optimization problems. *Soft Computing*, 18(7):1405–1424, 2013.