An Adaptive Local Search Algorithm for Real-Valued Dynamic Optimization

Michalis Mavrovouniotis*, Ferrante Neri*† and Shengxiang Yang*
*Centre for Computational Intelligence, School of Computer Science and Informatics, De Montfort University,
The Gateway, Leicester LE1 9BH, United Kingdom,
Email: {mmavrovouniotis@dmu.ac.uk, fneri@dmu.ac.uk, syang@dmu.ac.uk}
†Department of Mathematical Information Technology, University of Jyväskylä,
P.O. Box 35 (Agora), 40014 Finland

***Abstract*—This paper proposes a novel adaptive local search algorithm for tackling real-valued (or continuous) dynamic optimization problems. The proposed algorithm is a simple single-solution based metaheuristic that perturbs the variables separately to select the search direction for the following step and adapts its step size to the gradient. The search directions that appear to be the most promising are rewarded by a step size increase while the unsuccessful moves attempt to reverse the search direction with a reduced step size. When the environment is subject to changes, a new solution is sampled and crosses over the best solution in the previous environment. Furthermore, the algorithm makes use of a small archive where the best solutions are saved. Experimental results show that the proposed algorithm, despite its simplicity, is competitive with complex population-based algorithms for tested dynamic optimization problems.**

I. Introduction

Deterministic or randomized local search is sometimes a simple and yet powerful tool to tackle some optimization problems. Moreover, local search can be one very appealing option for those engineering problems characterized by a modest hardware, see [8]. Efficient local search algorithms, if combined with other algorithms or within a simple restarting mechanism, are proven to offer a performance competitive with that of complex metaheuristics. One extremely simple and successful example of local search component is the so called "S" algorithm. First introduced within a framework for large scale optimization in [27] and then readjusted in [15] with the name "Short Distance Exploration", or simply "S", this operator is a powerful steepest-descent method employing a perturbation logic derived from Hook-Jeeves [14]. This local search operator is also used within a multiple local search framework in [9] and [10], while in [8] S is implemented in a multi-start local search method. Recently, two variants of S, namely "S2" and "S3", have been proposed in [17]. The S3 algorithm is a simple adaptive version of S that performs moves along the axes by adjusting the step size to the successful directions, i.e., by following the gradient.

This paper proposes a variant of S3, denoted "DynS3", for solving continuous dynamic optimization problems (DOPs), where the fitness landscape changes over time. Many real-world problems have such characteristics that makes DOPs difficult problems to address. This is because the moving optimum of DOPs needs to be tracked. Particularly, once optimization algorithms converge to an optimum; they face a serious challenge to escape from it and capture the new optimum when a dynamic change occurs. Techniques that transfer knowledge from previous environments and maintain diversity are widely used to address DOPs [11], [13], [28]. Usually, these techniques are suitable only when the dynamic changes are of "small to medium" magnitudes [4].

The difference of the proposed DynS3 with the standard S3 lies in that the former algorithm incorporates an explicit memory (archive) to store the solutions found by the local search. DynS3 is applied to a single random solution whenever the computational budget is exhausted; and repeats its optimization task from a new random solution. When a dynamic change occurs, explicit actions are taken to speed up the re-optimization process by transferring knowledge from the archive. Hence, DynS3 starts optimizing from the best solution retrieved from the archive when a change occurs. A crowding strategy is used to update the archive in order to maintain diversity, which is an essential factor when addressing DOPs.

The experimental study of this paper is based on the moving peaks benchmark (MPB) problem [4] to evaluate the performance of the proposed local search algorithm in dynamic environments. The performance of the proposed DynS3 algorithm is compared with several state-of-the-art algorithms that were developed for DOPs in the literature. Although DynS3 is not incorporated with any global search or multi-search technique, the results show that the proposed algorithm is competitive with evolutionary algorithms (EAs) and particle swarm optimization (PSO) algorithms.

The rest of the paper is organized as follows. Section II describes the MPB which is the test suite used in the experiments. Section III describes some of the existing algorithms applied to DOPs. Section IV describes in detail the different components of the proposed DynS3 algorithm. Section V-C presents the experimental study and analyses the results. Finally, Section VI concludes this paper with relevant future works and directions.

II. Moving Peaks Benchmark Problem

Without loss of generality, in this paper we consider as our ultimate goal the maximization of an objective function $f(\mathbf{x})$, where $\mathbf{x} = [x_1, x_2, \ldots, x_d] \in \mathbb{R}^d$ is the decision vector and d is the number of variables (dimensions). The feasible solution space (or decision space $\mathbf{D}$) is constrained by the lower bound $\mathbf{L} = [l_1, l_2, \ldots, l_d]$ and the upper bound $\mathbf{H} = [h_1, h_2, \ldots, h_d]$

978-1-4799-7492-4/15/$31.00

of the decision vector $\mathbf{x}$. Since f is assumed to be a time-variant function, the time parameter t is introduced such that the objective function becomes $f(\mathbf{x}, t)$. Note that throughout this paper, arrays are highlighted in the bold face while scalars are in normal italic.

The MPB problem proposed by Branke [4] is one of the most widely accepted test suits for dynamically changing fitness landscapes. It has been widely used to analyse and compare algorithms in dynamic environments. The MPB problem is a maximization problem that consists of a number of peaks that randomly vary their shape (e.q., width and height) and position. Formally, the MPB function is defined as follows:

$$f(\mathbf{x}, t) = \max_{i=1,\ldots,m} \frac{H_i(t)}{1 + W_i(t) \sum_{j=1}^{d} (x_j(t) - X_{ij}(t))^2} \quad (1)$$

where $H_i(t)$ and $W_i(t)$ are the height and width of peak i at time t, respectively, and $X_{ij}(t)$ is the jth element of the location of peak i at time t. Every a function evaluations, the height and width of every peak i are modified by adding a normal distributed random number σ (with the mean 0 and variation 1) as follows:

$$H_i(t) = H_i(t-1) + height\ severity * \sigma \quad (2)$$

$$W_i(t) = W_i(t-1) + width\ severity * \sigma \quad (3)$$

Moreover, the location of every peak i is moved in a random direction by a vector $\mathbf{u_i}$ of a distance s as follows:

$$\mathbf{X_i}(t) = \mathbf{X_i}(t-1) + \mathbf{u_i} \quad (4)$$

where $\mathbf{u_i}$ is a linear combination of a random vector and the direction of the previous move defined as follows:

$$\mathbf{u_i}(t) = \frac{s}{|\mathbf{r} + \mathbf{u_i}(t-1)|}(1-\lambda)\mathbf{r} + \lambda \mathbf{u_i}(t-1)) \quad (5)$$

where $\mathbf{u_i}(t)$ is the shift vector, $\mathbf{r}$ is the random vector and λ is the correlation coefficient parameter. Note that a and s determine the frequency and magnitude of dynamic changes.

III. Previous Works

Most algorithms proposed to tackle the MPB problem are multi-population based EAs and PSO algorithms. Such algorithms are suitable for DOPs because they are inspired from nature, which is a continuous adaptation process [16], [23]. The first algorithm proposed to tackle the MPB problem is the memory-based multi-population EA proposed by Branke [4], where the best individuals are stored in an explicit memory. When a dynamic change occurs, the memory individuals are then merged to the evolving population. Another multi-population EA based on the forking mechanism, called the self organized scouts (SOS), was proposed in [5], [6]. This algorithm starts from a single population (i.e., parent) that explores the entire search space and allocates sub-populations (children) to promising areas to track local optima.

Blackwell and Branke [2], [3] introduced two multi-swarm algorithms, i.e., multi charged PSO (mCPSO) and multi-quantum swarm optimization (mQSO), inspired from the atom field. The former algorithm contains a sub-swarm of charged particles that repel each other and circle around another sub-swarm of neutral (conventional) particles. The latter algorithm is a variation of the former algorithm, where the charged particles move randomly within a fixed radius around the particle with the best fitness (i.e., attractor). Both algorithms use an exclusion radius to prevent sub-swarm to overlap.

Parrot and Li [24] proposed the speciation PSO (SPSO) where close particles are considered as members of the same sub-swarm (species). A fixed radius is defined to create the sub-swarms, where each sub-swarm has a maximum threshold of the swarm size. Whenever the number of members within a sub-swarm exceeds the predefined threshold, the worst members are reinitialized to random positions. Du and Li [12] proposed another multi-swarm PSO with two fixed sub-swarms, called multi-strategy ensemble PSO (MEPSO). One sub-swarm is responsible for diversification (exploration) whereas the other sub-swarm is responsible for intensification (exploitation).

Mendes and Mohais [20] introduced a multi-population dynamic differential evolution (DynDE) based on the exclusion concept described above. Within the DynDE, individuals are generated according to their types: named DE, entropy DE, Quantum or Brownian, where the last three types aim to maintain diversity. Lung and Dumitrescu [19] proposed a hybridized algorithm that combines PSO and crowding DE, called collaborative evolutionary-swarm optimization (CESO). Populations of the two methods collaborate to achieve a good balance between exploration and exploitation. The crowding DE is responsible to maintain diversity by replacing the closest individual only if it is fitter, whereas PSO is responsible to locate and converge to the global optimum. Whenever a dynamic change occurs, the PSO swarm is reinitialized to the crowding DE population.

Moser and Hendtlass [21] investigated the use of local search as a main feature to tackle DOPs by introducing the multi-phase multi-individual extremal optimization (MMEO) algorithm. The key idea of the MMEO algorithm is to stepwise sample every dimension in equal distances [1]. The extremal optimization algorithm is used to determine the initial solution for the local search procedure by selecting the best sample. The resulting solutions are stored in a memory that uses further local searches to track the local optima when a dynamic change occurs. Similarly, Lepagnot *et al.* [18] proposed a multi-agent algorithm for dynamic optimization (MADO) that follows a similar concept with MMEO. Their difference lies in that in MADO multiple local searches are performed by a population of agents.

IV. Proposed DynS3 Algorithm for Dynamic Optmization

A. The Framework of Dynamic S3 (DynS3)

The framework of DynS3 does not make use of any population-based algorithm or multi-local search technique. The DynS3 local search operator is iteratively applied to a single solution in order to locate and track the moving optimum. More precisely, DynS3 performs local search to a random solution; moving from its current position to a better one within its neighbourhood until the computational budget (e.g., a maximum number of trials) is exhausted or no better solution is available (hopefully reaching a local optimum).

Algorithm 1 DynS3 Framework

1: initialize archive A with random solutions
2: generate initial solution $\mathbf{x_0}$ randomly within $\mathbf{D}$
3: $\mathbf{x_{best}} = \emptyset$
4: **while** *stop condition not met* **do**
5: $\quad \mathbf{x_s} \leftarrow$ S3($\mathbf{x_0}$)
6: $\quad$ **if** $f(\mathbf{x_s}) > f(\mathbf{x_{best}})$ **then**
7: $\quad\quad \mathbf{x_{best}} \leftarrow \mathbf{x_s}$
8: $\quad$ **end if**
9: $\quad$ **if** *dynamic change is detected* **then**
10: $\quad\quad$ UpdateArchive($\mathbf{x_{best}}$)
11: $\quad\quad \mathbf{x_{mem}} \leftarrow$ retrieve best archived solution
12: $\quad\quad \mathbf{x_s} \leftarrow$ ExponentialCrossover($\mathbf{x_{mem}}$)
13: $\quad\quad \mathbf{x_0} \leftarrow \mathbf{x_s}$
14: $\quad$ **else**
15: $\quad\quad$ // check status of S3 local search
16: $\quad\quad$ **if** $\delta \leq \epsilon$, (suggested value: $\epsilon = 0.001$) **then**
17: $\quad\quad\quad$ reset δ values (using Algorithm 2 Lines 2–4)
18: $\quad\quad$ **end if**
19: $\quad\quad$ **if** *budget is exhausted*, (e.g., 50 trials) **then**
20: $\quad\quad\quad$ UpdateArchive($\mathbf{x_s}$)
21: $\quad\quad\quad$ generate a random solution $\mathbf{x_r}$ within $\mathbf{D}$
22: $\quad\quad\quad \mathbf{x_0} \leftarrow \mathbf{x_r}$
23: $\quad\quad$ **end if**
24: $\quad$ **end if**
25: **end while**

Algorithm 2 S3($\mathbf{x_0}$)

1: $\mathbf{x_s} \leftarrow \mathbf{x_0}$
2: **for** $i = 1 : d$ **do**
3: $\quad \delta[i] \leftarrow \alpha$ ($\alpha \in \left(0, \mathbf{x^U}[i] - \mathbf{x^L}[i]\right)$, suggested value: 1)
4: **end for**
5: **while** *stop condition not met* **do**
6: $\quad$ **for** $i = 1 : d$ **do**
7: $\quad\quad \mathbf{x_{trial}}[i] \leftarrow \mathbf{x_s}[i] - \delta[i]$
8: $\quad\quad$ **if** $f(\mathbf{x_{trial}}) \geq f(\mathbf{x_s})$ **then**
9: $\quad\quad\quad \mathbf{x_s} \leftarrow \mathbf{x_{trial}}$
10: $\quad\quad\quad \delta[i] \leftarrow \delta[i] * c_e$ ($c_e \in (1, 2]$, suggested value: 1.1)
11: $\quad\quad$ **else**
12: $\quad\quad\quad \delta[i] \leftarrow -\delta[i] * c_c$ ($c_c \in (0, 1]$, suggested value: 0.5)
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: **end while**
16: **Output** $\mathbf{x_s}$

The solutions found are stored in an archive for two main reasons: 1) to use a solution as a detector to detect an environmental change and 2) to use the best solution as the initial point for S3 local search to accelerate the re-optimization process when the environment changes. However, before the best archived solution is passed to the S3 for optimization to the new environment an exponential crossover is performed in order to help local search escape from a possible local optimum.

The pseudocode of the overall framework of DynS3 is described in Algorithm 1 and the components integrated are described in detail in the following subsections. Since DynS3 is integrated with the MPB problem the stop condition of the algorithm in line 4 is basically the maximum number of function evaluations allowed.

B. Local Search Strategy

The pseudocode of the S3 local search strategy is shown in Algorithm 2. The working of S3 is described as follows. Given an initial starting point $\mathbf{x_s}$, S3 perturbs it by performing asymmetric moves along both directions of each dimension. These moves are ruled by a vector of step-sizes δ, whose components for $i = 1, 2, \ldots d$ are indicated as $\delta[i]$. For each dimension i, the S3 algorithm at first attempts to calculate

$$\mathbf{x_s}[i] - \delta[i]. \tag{6}$$

If this move is successful, the perturbed solution $\mathbf{x_{trial}}$ replaces the solution $\mathbf{x_s}$ and the corresponding step-size $\delta[i]$ is expanded by means of a multiplication factor $c_e \in (1, 2]$. If, conversely, the move is unsuccessful, the step-size $\delta[i]$ is reduced by a contraction factor $c_c \in (0, 1]$ and the opposite direction is explored. Hence, S3 expands step-sizes to accelerate the search in fruitful directions and both toggles the sign and contracts step-sizes upon failing to yield an improvement.

Moreover, it must be remarked that S3 does not immediately evaluate an alternate solution upon failure, but defers the evaluation to the next iteration. In this manner, step sizes are gradually oriented towards improved fitness and this strategy avoids precomputing alternate solutions in directions known to be worse and thereby makes better use of the computational budget. Finally, as explained in [17], this step-size adaptation, despite its simplicity, appears to be effective at detecting the most promising search directions (along the variables) and thus tackling ill-conditioned optimization problems.

Fig. 1 illustrates the adaptive step size behaviour of S3. After a successful attempt in one direction (horizontal in the example), S3 increases the step size by a factor c_e. When the next move is unsuccessful, S3 reduces the step size by a factor c_c and attempts to perturb the variable in the opposite direction in the next round of local search.

C. Archive Maintenance

Every time S3 exceeds the given computational budget, it is restarted from a new random solution. The purpose of the archive A is to store and maintain the solutions found by the local search algorithm before it is restarted and the best so far solution just before an environmental change occurs. The size of A (denoted n_a) is fixed and initially A contains random solutions.

Every time a solution needs to be stored, the archive is updated (see Algorithm 3) as follows: the closest solution within A, i.e., $\mathbf{x_{closest}}$ is replaced if the fitness of the solution to be stored, i.e., $\mathbf{x_{best}}$ is better. Note that the $\mathbf{x}_{best}$ could be either the best solution found before restarting S3 from a new random solution or the best so far solution just before an environmental change (in case it is not maintained); see lines 11 and 21 in Algorithm 1. The distance $\xi(\mathbf{x_a}, \mathbf{x_s})$ between the two solutions $\mathbf{x_{closest}}$ and $\mathbf{x_{best}}$ in the d-dimension space is based on the Euclidean distance defined as follows:

$$\xi(\mathbf{x_{closest}}, \mathbf{x_{best}}) = \sqrt{\sum_{i=1}^{d} (\mathbf{x_{closest}}[i] - \mathbf{x_{best}}[i])^2} \tag{7}$$

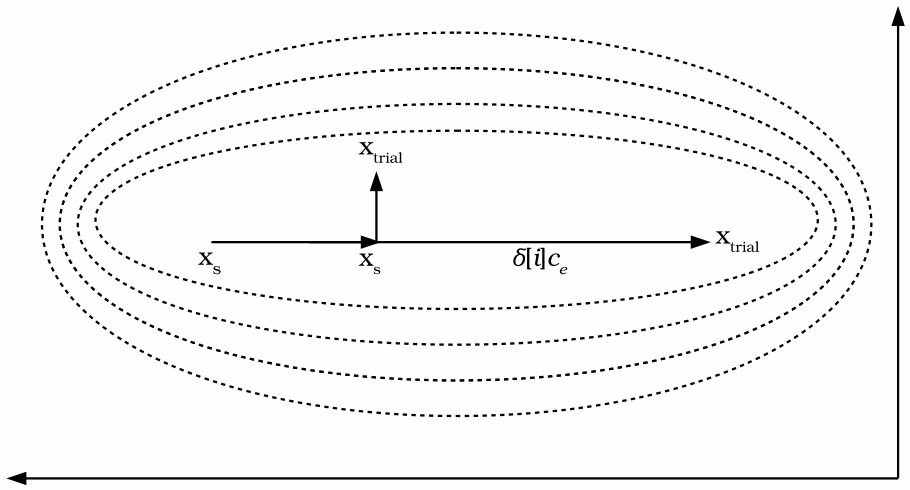

Fig. 1. Illustration of the S3 search logic.

Algorithm 3 UpdateArchive($\mathbf{x_{best}}$)

1: $j \leftarrow 0$
2: $\mathbf{x_{closest}} \leftarrow A[j]$
3: $min \leftarrow \xi(\mathbf{x_{closest}}, \mathbf{x_{best}})$
4: $i \leftarrow 1$
5: **while** $i < n_a$, (suggested value: $n_a = 10$) **do**
6: $\quad \mathbf{x_{closest}} \leftarrow A[i]$
7: $\quad$ **if** $min < \xi(\mathbf{x_{closest}}, \mathbf{x_{best}})$ **then**
8: $\quad\quad min \leftarrow \xi(\mathbf{x_{closest}}, \mathbf{x_{best}})$
9: $\quad\quad \mathbf{x_{closest}} \leftarrow \mathbf{x_{best}}$
10: $\quad\quad j \leftarrow i$
11: $\quad$ **end if**
12: $\quad i \leftarrow i + 1$
13: **end while**
14: **if** $f(\mathbf{x_{best}}) > f(\mathbf{x_{closest}})$ **then**
15: $\quad A[j] \leftarrow \mathbf{x_{best}}$
16: **end if**

Algorithm 4 ExponentialCrossover($\mathbf{x_{mem}}$)

1: $\mathbf{x_s} \leftarrow \mathbf{x_{mem}}$
2: generate a random solution $\mathbf{x_r}$ within $\mathbf{D}$
3: generate $i \leftarrow round\,(d * rand\,(0, 1))$
4: $\mathbf{x_r}[i] \leftarrow \mathbf{x_s}[i]$
5: **while** $rand\,(0, 1) \leq Cr$, (suggested value: $Cr = 0.3$) **do**
6: $\quad \mathbf{x_r}[i] \leftarrow \mathbf{x_s}[i]$
7: $\quad i \leftarrow i + 1$
8: $\quad$ **if** $i == d$ **then**
9: $\quad\quad i \leftarrow 1$
10: $\quad$ **end if**
11: **end while**
12: $\mathbf{x_s} \leftarrow \mathbf{x_r}$
13: **if** $f(\mathbf{x_{mem}}) > f(\mathbf{x_s})$ **then**
14: $\quad \mathbf{x_s} \leftarrow \mathbf{x_{mem}}$
15: **end if**
16: **Output** $\mathbf{x_s}$

where a value closer to 0 indicates that the two solutions are identical. This way, the archive eliminates duplicate solutions and maintains a diverse set of solutions that can be used when an environmental change occurs.

D. Detecting Environmental Changes

The detection of changes is important for algorithms in order to tackle efficiently dynamic environments [26]. This is because explicit actions need to be taken to help the optimization process escape from the previous optimum and locate the new one.

A change in the fitness landscape on the MPB problem will affect the solutions stored in the archive. Based on this fact, the detection of an environmental change is straightforward: a single solution stored in the archive is re-evaluated. If there is a change in the fitness, then a dynamic change occurred. Note that, whenever a dynamic change occurs, all the solutions in the archive are re-evaluated to be compatible with the new environment.

E. Handling Environmental Changes

Every time an environmental change is detected, a new solution $\mathbf{x_r}$ is sampled at random within the decision space $\mathbf{D}$. Then, an exponential crossover in a DE fashion, e.g., see [25], is performed between $\mathbf{x_r}$ and the best solution from the archive $\mathbf{x_s}$. This crossover occurs in the following way. One variable from $\mathbf{x_s}$ is randomly selected. Note that the $round$ function in line 3 rounds the number to its nearest integer value. This variable replaces the corresponding variable within the solution $\mathbf{x_r}$. Then, a set of random numbers between 0 and 1 are generated. As long as $rand\,(0, 1) \leq Cr$, where the crossover rate Cr is a predetermined parameter, the design variables from the solution $\mathbf{x_s}$ are copied into the corresponding positions of the random solution $\mathbf{x_r}$. The first time that $rand\,(0, 1) > Cr$, the copy process is interrupted. The newly generated solution is then used for the subsequent S3 activation in the new environment, i.e., $\mathbf{x_s}$ is used for the new environment. The pseudocode of the exponential crossover is shown in Algorithm 4.

According to our interpretation, this operation allows to

TABLE I. DEFAULT SETTINGS FOR SCENARIO 2 OF THE MOVING PEAKS BENCHMARK

Parameter	Value
Number of peaks (m)	10
Dimension (d)	5
Peak heights (H)	$[30.0, 70.0]$
Peak widths (W)	$[1, 12]$
Peak shape	cone
Basic function	false
Frequency of change (a)	5000 evaluations
Severity of change (s)	1.0
Height severity	7.0
Width severity	1.0
Correlation coefficient (λ)	0.0
Number of changes (T)	100

partially use the quality of the solution in the previous environment combined with a certain degree of randomization. The new environment/scenario is expected to be different from the previous one, but still related to it. This would be the situation also in practical problems where a change varies the optimization problem without a dramatic variation of the problem features. Moreover, due to its inner structure, the exponential crossover maintains sections of the solution thus displaying a high exploitation of the search, see [22] and [15]. Hence, the new individual can be seen as the best solution found in the previous environment after a perturbation that modifies a section of the solution but partially maintains the structure of the previous best solution $\mathbf{x_s}$.

V. EXPERIMENTAL STUDY

A. Experimental Setup

Two sets of experiments were carried out based on the MPB problem [4] described previously. The objective of the first set of experiments is to investigate the different components of the proposed DynS3 algorithm (see Tables II). In the second set of experiments, the performance of DynS3 is compared with a number of peer algorithms taken from the literature. The selected algorithms include MMEO [21], CESO [19], MADO [18], mQSO [2], mCPSO [2], MEPSO [12] and SPSO [24]. Note that the results reported in this paper for these algorithms are the ones reported in their original papers since they were performed on the same MPB problem. For some cases, the results are not reported and they are indicated as "n/a" in the tables (see Tables III, IV and V later on).

Three sets of parameters (i.e., three scenarios) were proposed for the MPB problem in [4]. The most commonly used set of parameters appears to be scenario 2, and thus, it is used in this paper. The default settings for scenario 2 are given in Table I. From Table I, it can be observed that each algorithm performs 5×10^5 evaluations (e.g., $T \times a$). When dealing with dynamic environment, it is not useful to simply compare the best so far solution because the global optimum is changing over time [16]. Hence, the modified offline error is used to evaluate the performance of the algorithms, which averages over the best so far solution found since the last dynamic change [6]. The modified offline error is defined as follows:

$$E_{offline} = \frac{1}{T}\sum_{i=1}^{T}\left(\frac{1}{a}\sum_{j=1}^{a} e^*_{ij}\right) \quad (8)$$

TABLE II. TRIALS WITH AND WITHOUT INDIVIDUAL COMPONENTS

Variation description	Offline Error $\pm$ Standard Error
DynS3	2.32 ± 0.01
DynS3$_{NoCrossover}$	2.47 ± 0.01
DynS3$_{NoArchive}$	4.40 ± 0.04
DynS3$_{NoDetection}$	10.29 ± 0.09
DynS1	7.72 ± 0.25
DynS2	7.70 ± 0.23

where T and a are defined in Table I, e^*_{ij} is the difference (error) between the value of the optimal solution for the i-th environment and the value of the best solution found by the algorithm since the last dynamic change. All reported results are averaged over 50 runs with different random seeds for the algorithm but on the same MPB function.

All the results are expressed in terms of average over the performed runs and corresponding standard deviation. Furthermore, statistical comparisons of two-tailed t-Test with 0.05 confidence level have been performed. The results of the t-Test are also reported next to each average fitness $\pm$ standard deviation. The statistical outperformance of the proposed DynS3 with respect to each competitor is indicated with a "$+$". On the contrary "$-$" indicates that DynS3 is outperformed.

B. Results of Analysing Different Components in DynS3

Since S3 was proposed for stationary environments, many components were combined with S3, i.e., exponential crossover, dynamic change detection mechanism and an archive, to make it sufficient for dynamic environments. The experiments in this section report the contribution, regarding the offline error, for each individual component within DynS3.

DynS3 is the proposed algorithm with all the components. DynS3$_{NoCrossover}$ omits the exponential crossover. Hence, the best solution selected from the archive is not modified. DynS3$_{NoArchive}$ does not maintain an archive of the best solutions found by S3. This means that when a dynamic change occurs the algorithm is restricted to select only the best solution of the previous environment and use it for the re-optimization process. Note that exponential crossover is also performed in the DynS3$_{NoArchive}$ variation. DynS3$_{NoDetection}$ does not contain any mechanism to detect a dynamic change. Therefore, the archive is not used because no solution is used to re-optimize when a dynamic change occurs.

Furthermore, DynS3 is compared with the dynamic variants of two other local search algorithms, denoted DynS1 and DynS2, respectively, which are local search algorithms S and S2 reported in [17] endowed with the mechanisms for detecting and handling the environmental changes as well as the archive used in DynS3. In other words, DynS1, DynS2, and DynS3 work with the same components for dynamic problems but different search logic. The DynS1 algorithm does not expands and contract the step sizes (see also [9]) while DynS2 checks each variable along both the directions before selecting the new $\mathbf{x_s}$, see [17] for details.

Table II summarizes the results of DynS3 against its variants mentioned above as well as against DynS1 and DynS2 on the MPB for scenario 2 (see Table I). Furthermore, Fig. 2

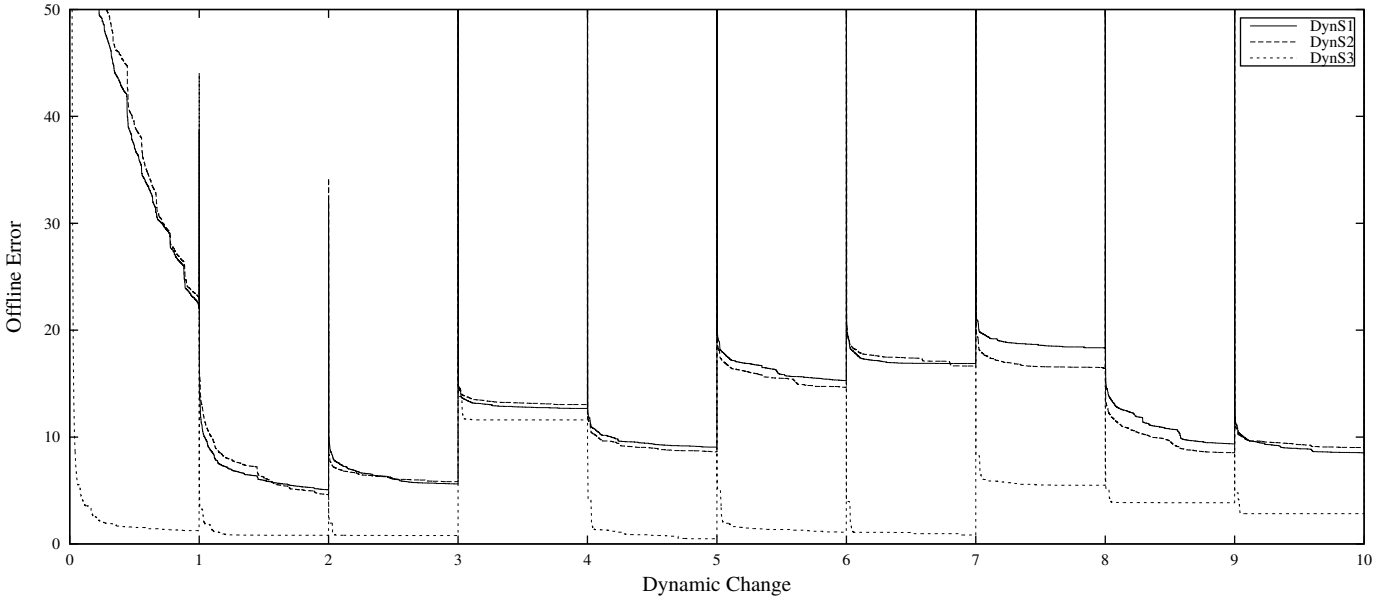

Fig. 2. Dynamic performance of local search algorithms.

presents the dynamic behaviour of DynS1, DynS2 and DynS3 on the first 10 environments.

From Table II, it can be observed that, when the detection mechanism is omitted, the performance of DynS3$_{NoDetection}$ is degraded significantly when compared with the performance of DynS3. This is natural because DynS3$_{NoDetection}$ is basically the plain S3 with restart designed to tackle static environments. Hence, when a dynamic change occurs, the algorithm may continue optimizing to a non-promising area until it restarts from a new random solution. In addition, the S3 algorithm is always restarted from a random solution since knowledge from previous environments is not performed. The archive component is also important since the performance of DynS3$_{NoArchive}$ is also degraded significantly (but not to the level of DynS3$_{NoDetection}$). It is not always the case that the best so far solution of the previous environment will be still quite fit for reusing for the new environment. Hence, maintaining an archive of promising solutions may increase the chances of choosing a fit solution. In contrast, the exponential crossover is slightly affecting the performance because the performance of DynS3$_{NoCrossover}$ is slightly degraded when compared with the performance of DynS3.

When other local search strategies are used the performance is also degraded. This is supported from the offline error results of DynS1 and DynS2 when compared with DynS3. From Fig. 2, it can be observed that DynS3 locates the global optimum faster than its competitors. DynS1 and DynS2 have a similar behaviour and need more time to locate the optimum which can be observed from the behaviour of the initial environment. Hence, DynS1 and DynS2 are unable to store fit solutions to the archive and negatively affect their performance on the next environments. In contrast, DynS3 usually maintains very low offline error during the environmental changes (except when the third dynamic change occurs). These results conform the comparison study reported in [17], which has shown how the adaptive step logic is effective in diverse problem conditions.

TABLE III. COMPARISON WITH OTHER ALGORITHMS ON THE MOVING PEAKS BENCHMARK WITH THE DEFAULT SETTINGS (TABLE I) AND DIFFERENT d VALUES

Algorithm	Offline Error $\pm$ Standard Error (t-Test result)	
$d \Rightarrow$	5	100
MADO [18]	0.59 ± 0.10 $-$	34.64 ± 2.72 $+$
MMEO [21]	0.66 ± 0.20 $-$	480.5 ± 70.1 $+$
CESO [19]	1.38 ± 0.02 $-$	24.60 ± 0.25 $-$
mQSO [2]	1.72 ± 0.06 $-$	n/a
mCPSO [2]	2.05 ± 0.07 $-$	n/a
DynS3	2.32 ± 0.01	28.04 ± 0.10
SPSO [24]	2.51 ± 0.09 $+$	n/a
MEPSO [12]	4.02 ± 0.56 $+$	n/a

C. Results on Comparing DynS3 with Peer Algorithms

1) Varying the Number of Dimensions: In Table III, the results of the proposed DynS3 algorithm compared with other peer algorithms are presented for 5 (default) and 100 dimensions. The higher the dimensionality of the problem, the more increasing complexity the problem becomes. Therefore, it is straightforward that the offline error of the algorithms increases significantly from 5 to 100 dimensions.

DynS3 performs better than SPSO and MEPSO under 5 dimensions whereas it is outperformed by the remaining algorithms. The best results under 5 dimensions are achieved by MADO and MMEO. However, DynS3 outperforms these algorithms under 100 dimensions.

2) Varying the Severity of Change: In the default settings, the severity of change was set to 1. Here, we vary the value from 1 to 6 to further investigate the performance of the algorithms. The results with respect to the offline error are presented in Table IV.

TABLE IV. COMPARISON WITH OTHER ALGORITHMS ON THE MOVING PEAKS BENCHMARK WITH THE DEFAULT SETTINGS (TABLE I) AND DIFFERENT s VALUES

Algorithm	Offline Error $\pm$ Standard Error (t-Test result)					
$s \Rightarrow$	1.0	2.0	3.0	4.0	5.0	6.0
MADO [18]	0.59 ± 0.10 $-$	0.87 ± 0.12 $-$	1.18 ± 0.13 $-$	1.49 ± 0.13 $-$	1.86 ± 0.17 $-$	2.32 ± 0.18 $-$
MMEO [21]	0.66 ± 0.20 $-$	0.86 ± 0.21 $-$	0.94 ± 0.22 $-$	0.97 ± 0.21 $-$	1.05 ± 0.21 $-$	1.09 ± 0.22 $-$
CESO [19]	1.38 ± 0.02 $-$	1.78 ± 0.02 $-$	2.03 ± 0.03 $-$	2.23 ± 0.05 $-$	2.52 ± 0.06 $-$	2.74 ± 0.10 $-$
mQSO [2]	1.72 ± 0.06 $-$	2.40 ± 0.06 $-$	3.00 ± 0.06 $-$	3.05 ± 0.10 $-$	4.24 ± 0.10 $-$	4.79 ± 0.10 $-$
mCPSO [2]	2.05 ± 0.07 $-$	2.80 ± 0.07 $-$	3.57 ± 0.08 $+$	4.18 ± 0.09 $+$	4.89 ± 0.11 $+$	5.53 ± 0.13 $+$
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
SPSO [24]	2.51 ± 0.09 $+$	3.78 ± 0.09 $+$	4.96 ± 0.12 $+$	5.65 ± 0.13 $+$	6.76 ± 0.15 $+$	7.68 ± 0.16 $+$
MEPSO [12]	4.02 ± 0.56 $+$	n/a	n/a	n/a	n/a	n/a

TABLE V. COMPARISON WITH OTHER ALGORITHMS ON THE MOVING PEAKS BENCHMARK WITH THE DEFAULT SETTINGS (TABLE I) AND DIFFERENT m VALUES

Algorithm	Offline Error $\pm$ Standard Error (t-Test result)							
$m \Rightarrow$	1	10	20	30	40	50	100	200
MADO [18]	n/a	0.59 ± 0.10 $-$	n/a	n/a	n/a	n/a	n/a	n/a
MMEO [21]	n/a	0.66 ± 0.20 $-$	n/a	n/a	n/a	n/a	n/a	n/a
CESO [19]	1.04 ± 0.00 $+$	1.38 ± 0.02 $-$	1.72 ± 0.02 $-$	1.24 ± 0.01 $-$	1.30 ± 0.02 $-$	1.45 ± 0.01 $-$	1.28 ± 0.02 $-$	n/a
mQSO [2]	5.07 ± 0.17 $+$	1.72 ± 0.06 $-$	2.74 ± 0.07 $+$	3.27 ± 0.11 $+$	3.60 ± 0.08 $+$	3.65 ± 0.11 $+$	3.93 ± 0.08 $+$	3.86 ± 0.07 $-$
mCPSO [2]	4.93 ± 0.17 $+$	2.05 ± 0.07 $-$	2.95 ± 0.08 $+$	3.38 ± 0.11 $+$	3.69 ± 0.11 $+$	3.68 ± 0.11 $+$	4.07 ± 0.09 $+$	3.97 ± 0.08 $+$
DynS3	0.22 ± 0.00	2.32 ± 0.01	2.65 ± 0.03	3.03 ± 0.03	2.88 ± 0.05	3.27 ± 0.02	1.98 ± 0.01	2.11 ± 0.01
SPSO [24]	2.64 ± 0.10 $+$	2.51 ± 0.09 $+$	3.21 ± 0.07 $+$	3.64 ± 0.07 $+$	3.85 ± 0.08 $+$	3.86 ± 0.08 $+$	4.01 ± 0.07 $+$	3.82 ± 0.05 $+$
MEPSO [12]	0.53 ± 0.15 $+$	4.02 ± 0.56 $+$	4.19 ± 0.57 $+$	4.27 ± 0.83 $+$	n/a	4.20 ± 0.47 $+$	n/a	n/a

Generally, as the severity increases the offline error of the algorithm increases. This is natural because the higher the s value, the more severe the change, and thus, the more difficult the problem becomes to address. The rankings regarding the performance of the algorithms are similar with when the default values ($s = 1$) are set.

3) Varying the Number of Peaks: In the default settings, the number of peaks was set to 10. Obviously, it is easier to find the global optimum when the number of peaks is smaller. Hence, we vary the number to 1, 20, 30, 40, 50, 100 and 200 to further investigate the performance of the algorithms. The results with respect to the offline error are presented in Table V.

For DynS3, the best results are obtained when a single peak exists, whereas achieves the second best results (following CESO) when the number of peaks is more than 10. MADO and MMEO have the best results when the number of peaks is 10 followed by CESO. Although DynS3 is beaten by mCPSO and mQSO when the number of peaks is 10, it beats them when a larger number of peaks are present.

In summary, it should be appreciated that DynS3 is a fairly simple light component in terms of the complexity of the operator and memory employment especially when compared with complex population based schemes. Despite its simplicity, DynS3 appears to display a respectable performance on a range of dynamic problems. Thanks to these features, DynS3 can be seen as a simple but yet efficient operator to be used as stand alone when the hardware limitations impose it such as in a robot control card, see [22], or integrated within a more complex framework when a high performance is the main priority of the problem. In the light of the latter category, DynS3 can be seen as a heuristic within a hyper-heuristic, see [7], or as a local search operator within a memetic computing framework, see [10].

VI. CONCLUSIONS

This paper proposes a local search algorithm, DynS3, for handling real-valued dynamic environments. The proposed algorithm performs moves along the axes and incorporates a simple adaptive mechanism to quickly exploit the most promising search directions. In addition, DynS3 makes use of a re-sampling mechanism when an environment change is detected. This re-sampling generates a new initial solution by partially using the information on the best solution within the previous environment. Furthermore, DynS3 uses a memory archive to ensure that the best solutions are stored before the environmental change and maintain the diversity.

The proposed algorithm is relatively simple and still quite effective when compared to some complex population-based algorithms representing the state-of-the-art in dynamic optimization. The efficiency of DynS3 is especially evident when the environment contains many moving peaks.

This study can be seen as a preliminary work towards the harmonic inclusion of multiple search algorithms within a framework for dynamic optimization. Ideally, the coordination of the components would occur according to an automatic mechanism able to detect the proper algorithmic components to tackle the specific problem features. The proposed local search algorithm can be effectively used in those real-world applications where the hardware available imposes the use of a simple algorithm, especially in terms of memory employment or within a more complex framework that balances global and local search.

ACKNOWLEDGMENT

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/K001310/1.

REFERENCES

[1] P. Bak and K. Sneppen, "Punctuated equilibrium and criticality in a simple model of evolution," *Phys. Rev. Lett.*, vol. 71, no. 24, pp. 4083–4086, Dec 1993.

[2] T. Blackwell and J. Branke, "Multiswarms, exclusion, and anti-convergence in dynamic environments," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 459–472, Aug 2006.

[3] ——, "Multi-swarm optimization in dynamic environments," in *Applications of Evolutionary Computing*, ser. Lecture Notes in Computer Science, G. Raidl, S. Cagnoni, J. Branke, D. Corne, R. Drechsler, Y. Jin, C. Johnson, P. Machado, E. Marchiori, F. Rothlauf, G. Smith, and G. Squillero, Eds. Springer Berlin Heidelberg, 2004, vol. 3005, pp. 489–500.

[4] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, vol. 3, 1999, pp. 1875–1882.

[5] J. Branke, T. Kaussler, C. Smidt, and H. Schmeck, "A multi-population approach to dynamic optimization problems," in *Evolutionary Design and Manufacture*, I. Parmee, Ed. Springer London, 2000, pp. 299–307.

[6] J. Branke and H. Schmeck, "Designing evolutionary algorithms for dynamic optimization problems," in *Advances in Evolutionary Computing*, ser. Natural Computing Series, A. Ghosh and S. Tsutsui, Eds. Springer Berlin Heidelberg, 2003, pp. 239–262.

[7] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. Woodward, "Classification of hyper-heuristic approaches," in *Handbook of Meta-Heuristics*. Springer, 2010, pp. 449–468.

[8] F. Caraffini, F. Neri, B. Passow, and G. Iacca, "Re-sampled inheritance search: High performance despite the simplicity," *Soft Computing*, vol. 17, no. 12, pp. 2235–2256, 2014.

[9] F. Caraffini, F. Neri, G. Iacca, and A. Mol, "Parallel memetic structures," *Information Sciences*, vol. 227, no. 0, pp. 60 – 82, 2013.

[10] F. Caraffini, F. Neri, and L. Picinali, "An analysis on separability for memetic computing automatic design," *Information Sciences*, vol. 265, pp. 1–22, 2014.

[11] H. G. Cobb and J. J. Grefenstette, "Genetic algorithms for tracking changing environments," in *Proceedings of the 5th Int. Conf. Genetic Algorithms*, 1993, pp. 523–530.

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

[13] J. J. Grefenstette, "Genetic algorithms for changing environments," in *Proceedings of the 2nd Int. Conf. Parallel Problem Solving From Nature*, 1992, pp. 137–144.

[14] R. Hooke and T. A. Jeeves, "Direct search solution of numerical and statistical problems," *Journal of the ACM*, vol. 8, pp. 212–229, Mar. 1961.

[15] G. Iacca, F. Neri, E. Mininno, Y. S. Ong, and M. H. Lim, "Ockham's Razor in Memetic Computing: Three Stage Optimal Memetic Exploration," *Information Sciences*, vol. 188, pp. 17–43, 2012.

[16] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-a survey," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, June 2005.

[17] N. Khan and F. Neri, "Two local search components that move along the axes for memetic computing frameworks," in *Proceedings of the IEEE Symposium Series on Computational Intelligence*, 2014.

[18] J. Lepagnot, A. Nakib, H. Oulhadj, and P. Siarry, "Performance analysis of mado dynamic optimization algorithm," in *Proceedings of the 9th Int. Conf. on Intelligent Systems Design and Applications*, 2009, pp. 37–42.

[19] R. Lung and D. Dumitrescu, "Collaborative evolutionary swarm optimization with a gauss chaotic sequence generator," in *Innovations in Hybrid Intelligent Systems*, ser. Advances in Soft Computing, E. Corchado, J. Corchado, and A. Abraham, Eds. Springer Berlin Heidelberg, 2007, vol. 44, pp. 207–214.

[20] R. Mendes and A. Mohais, "Dynde: a differential evolution for dynamic optimization problems," in *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, vol. 3, 2005, pp. 2808–2815.

[21] 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*, 2007, pp. 252–259.

[22] F. Neri, G. Iacca, and E. Mininno, "Disturbed Exploitation compact Differential Evolution for Limited Memory Optimization Problems," *Information Sciences*, vol. 181, no. 12, pp. 2469–2487, 2011.

[23] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 6, pp. 1–24, Oct 2012.

[24] D. Parrott and X. Li, "Locating and tracking multiple dynamic optima by a particle swarm model using speciation," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 440–458, Aug 2006.

[25] K. V. Price, R. Storn, and J. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*. Springer, 2005.

[26] H. Richter, "Detecting change in dynamic fitness landscapes," in *Proceedings of the 2009 IEEE Congress on Evolutionary Computation*, 2009, pp. 1613–1620.

[27] L.-Y. Tseng and C. Chen, "Multiple trajectory search for Large Scale Global Optimization," in *Proceedings of the 2008 IEEE Congress on Evolutionary Computation*, 2008, pp. 3052–3059.

[28] S. Yang, "Genetic algorithms with memory- and elitism-based immigrants in dynamic environments," *Evolutionary Computation*, vol. 16, no. 3, pp. 385–416, 2008.