Robust optimization over time by learning problem space characteristics

Danial Yazdani, Trung Thanh Nguyen, and Jürgen Branke

Abstract—**Robust optimization over time is a new way to tackle dynamic optimization problems where the goal is to find solutions that remain acceptable over an extended period of time. The state-of-the-art methods in this domain try to identify robust solutions based on their future predicted fitness values. However, predicting future fitness values is difficult and error prone. In this paper, we propose a new framework based on a multi-population method in which sub-populations are responsible for tracking peaks and also gathering characteristic information about them. When the quality of the current robust solution falls below the acceptance threshold, the algorithm chooses the next robust solution based on the collected information. We propose four different strategies to select the next solution. The experimental results on benchmark problems show that our newly proposed methods perform significantly better than existing algorithms.**

Index Terms—**Dynamic optimization problems, Robust optimization over time, Tracking moving optima, Particle Swarm Optimization**

I. Introduction

MANY real-world optimization problems are dynamic and changing over time. Most previous studies in this domain focus on tracking the moving optimum (TMO) [1]. However, this is not practical in many real-world cases since changing solutions may be very costly, and changing the solution frequently is not desirable. For example, in scheduling, changing the schedule may have significant impact on suppliers and customers, or, in the design of telephone networks, sending out engineers to change the physical infrastructure can be very expensive. In taking-off/landing scheduling problem, it is desirable to keep the current implemented solution/schedule after an environmental change [2], [3] to avoid unfavorable disruptions in airport operations. To address such a problem, [4] proposed a new approach for solving dynamic optimization problems (DOP): finding solutions that are robust over the course of time. A robust solution is one that is not necessarily the best in the current environment, but that remains acceptable over several environments. A found robust solution can be utilized until its quality degrades to an unacceptable level.

In case the current robust solution becomes unsatisfactory, a new robust solution must be chosen. The process of finding a sequence of robust solutions is referred to as robust optimization over time (ROOT) [4], [5]. A DOP can be defined as:

$$F(\mathbf{x}) = f(\mathbf{x}, \theta^{(t)}), \tag{1}$$

where f is the objective function, $\mathbf{x}$ is a design vector, $\theta^{(t)}$ is environmental parameters which change over time and t is the time index with $t \in [0, T]$ where T is the problem life cycle or number of environments. In this paper, like most previous studies in the DOP domain, we investigate DOPs with $\theta^{(t)}$ that changes discretely. In this type of DOP, the environmental parameters change over time with stationary periods between changes. As a result, for a DOP with T environmental changes, we have a sequence of T static environments that can be described as:

$$F(\mathbf{x}) = \left[f(\mathbf{x}, \theta^{(1)}), f(\mathbf{x}, \theta^{(2)}), \ldots, f(\mathbf{x}, \theta^{(T)})\right], \tag{2}$$

where $\theta^{(i)}$ represents the environmental parameters in the i^{th} environment. For ROOT, the main goal is to minimize the number of times the chosen solution has to be changed because its performance drops below an acceptable level, or to maximize the average number of environments that a robust solution remains acceptable. Thus, the best case is that the first robust solution remains acceptable for all of the T environments and the worst case is that the number of robust solutions is equal to the number of environments (none of the solutions remained acceptable after even a single environmental change).

In this paper, we propose a new framework for ROOT. Its novelty and contribution are as follows. First, this framework uses multi-population methods to track and monitor each peak and learn about their characteristics. Second, in contrast to previous state-of-the-art frameworks which are based on predicting future fitness values of solutions, the proposed framework tries to predict future behaviors of peaks and then selects the next robust solution based on this information. Third, we propose four new strategies to select the next robust solution. Finally, we empirically evaluate our framework on a wide range of problem settings (different dimensions, change frequencies, shift severities and number of peaks), providing a detailed analysis on the performance of our new framework and demonstrating that it achieves better results than current state-of-the-art ROOT algorithms.

This work was supported in part by a Deans Scholarship by Faculty of Engineering and Technology, LJMU, a Newton Institutional Links grant no. 172734213, funded by the UK BEIS and delivered by the British Council, and a NRCP grant no. NRCP1617-6-125 delivered by Royal Academy of Engineering.

D. Yazdani is with the Liverpool Logistics, Offshore and Marine Research Institute, Department of Maritime and Mechanical Engineering, Liverpool John Moores University, Liverpool L3 3AF, United Kingdom (Email:danial.yazdani@gmail.com, d.yazdani@2016.ljmu.ac.uk).

T. T. Nguyen is with the Department of Maritime and Mechanical Engineering, Liverpool John Moores University, Liverpool L3 3AF, United Kingdom (Email:T.T.Nguyen@ljmu.ac.uk).

J. Branke is with the Operational Research and Management Sciences Group in Warwick Business school, University of Warwick, Coventry CV4 7AL, United Kingdom (Email:Juergen.Branke@wbs.ac.uk).

The rest of this paper is structured as follows. In Section II, related works are reviewed. In Section III, the proposed framework is presented. Section IV explains the experimental setup including benchmarks, performance indicators, comparison algorithms and parameter settings of all tested methods. Experimental results, analyses and comparisons with previous works are reported in Section V. In the final section, we summarize the main findings and suggest directions for future work.

II. Related works

The proposed framework uses multi-population or multi-swarm methods for ROOT. Therefore, in this section, we provide a more detailed literature review on ROOT as the main topic of this paper, as well as a brief review on multi-population/multi-swarm methods.

A. Robust optimization over time

In [4], ROOT was proposed as a new perspective on DOPs. A new framework for ROOT was proposed in [6] with the algorithm searching for robust solutions by means of local fitness approximation and prediction. This method consists of a population-based optimization algorithm, a fitness approximator (to estimate fitness at any point in the search space), a fitness predictor (to predict future fitness values) and a database. In [6], an adapted radial-basis-function network (RBFN) is the local approximator and an autoregressive (AR) is the predictor. A database was used for storing, in each iteration, all of the individuals' positions alongside their fitness values and the associated time of storage. This database was then used for approximating fitness values of solutions in previous environments which in turn was used for training the predictor.

It is important to have sufficient samples across the search space in this database to maintain the accuracy of the approximation. Since optimization algorithms quickly converge to the most promising region in the search space, there would be regions that they would not visit (such as regions with bad fitness) which still are necessary for having a good training data. On the one hand, in [6], the algorithm needs to have enough information to be able to predict any solution in the search space which is depended on the approximator. On the other hand, for having an appropriate approximation model, the training data needs to be properly distributed in the search space. For achieving this, in [6], authors generate half of the population using a specific hypercube design after each environmental change. Therefore, for each environment, the database can contain at least one solution from each hypercube. However, in larger environments such as ones with bigger search range and higher dimensions, the number of these hypercubes increases exponentially and becomes larger than the population size. As a result, the algorithm needs to evaluate a solution for each hypercube for adding to the database. These additional fitness evaluations become a challenge in larger problems.

To select a robust solution, [6] uses the sum of the solutions' current fitness value, its p previous fitness values (provided by the approximator) and its q future fitness values (provided by the predictor):

$$F(\mathbf{x}) = \sum_{l=t-p}^{t+q} f(\mathbf{x}, \theta^{(l)}), \tag{3}$$

where F is the sum of fitness values of $\mathbf{x}$ at time t. The performance of the proposed method in [6] depends on the accuracy of the approximation and prediction methods. In [6], a particle swarm optimization (PSO) [7] was used as the optimizer. In addition, several performance indicators were proposed, of which one of the most important ones is E_{avg}, the average error of the robust solution sequence $S = (\mathbf{r}_1, \mathbf{r}_2, \cdots, \mathbf{r}_k)$,

$$E_{\text{avg}} = \frac{1}{k}\sum_{i=1}^{k} e_i, \tag{4}$$

where

$$e_i = \frac{1}{n_i}\sum_{j=t_i}^{t_i+n_i-1} \left|\text{opt}^{(j)} - f(\mathbf{r}_i, \theta^{(j)})\right|, \tag{5}$$

$\text{opt}^{(j)}$ is the optimum fitness value at the j^{th} environment, $\mathbf{r}_i$ is the i^{th} robust solution, n_i is number of environments for which $\mathbf{r}_i$ remained acceptable, and t_i is the time that $\mathbf{r}_i$ was chosen. In other words, E_{avg} is the average error of robust solutions over all environments. Another performance indicator is ρ, the robustness rate of the robust solution sequence,

$$\rho = 1 - \frac{k-1}{T-1}, \tag{6}$$

where k is the number of robust solutions. In (6), a smaller k causes ρ to increase and the ideal situation happens when the first robust solution can remain acceptable in all environments i.e. $k = 1$. In addition, a new condition for checking whether a robust solution may be kept in a new environment was introduced. According to this condition, given a user defined threshold δ_{drop}, a robust solution $\mathbf{r}_i$ may be kept in the j^{th} environment if:

$$\left|\frac{f(\mathbf{r}_i, \theta^{(j)}) - \text{opt}^{(j)}}{\text{opt}^{(j)}}\right| \le \delta_{\text{drop}}. \tag{7}$$

In [8], authors proposed two new robustness definitions and metrics, namely *survival time* and *average fitness*. The survival time is the maximum time interval starting from time t during which the fitness value of the robust solution remains acceptable:

$$\mathcal{S}\left(\mathbf{x}, \theta^{(t)}, V\right) = \begin{cases} 0 & \text{if } f(\mathbf{x}, \theta^{(t)}) < V \\ 1 + \max\{l \mid \forall i \in \{t, \ldots, t+l\} : f\left(\mathbf{x}, \theta^{(i)}\right) \ge V\} & \text{otherwise} \end{cases} \tag{8}$$

where V is a user defined threshold. In (8), for each environment, $\mathcal{S}$ shows for how many environments the fitness value of the current solution has remained above V. Note the threshold V in [8] is easier to use than δ_{drop} in (7) from [6] which requires to know the optimum.

The robust solution is selected based on the predicted average fitness over a pre-defined time window ω as follows:

$$\mathcal{A}\left(\mathbf{x}, \theta^{(t)}, \omega\right) = \frac{1}{\omega}\sum_{i=0}^{\omega-1} f\left(\mathbf{x}, \theta^{(t+i)}\right). \tag{9}$$

When (8) and (9) are used as metrics, $f(\mathbf{x}, \theta^{(i)})$ for $i > t$ is predicted fitness value of $\mathbf{x}$ in t^{th} environment instead of its actual fitness value. In the experiments in [8], the authors assumed that the algorithm benefited from an ideal approximator so they used true previous fitness values instead of approximated fitness values. Consequently, the reported results in [8] were not subject to approximation errors. Additionally, a ROOT performance indicator was proposed in this paper based on (8) and (9) as follows:

$$\mathcal{P} = \frac{1}{T}\sum_{i=1}^{T} \mathcal{F}(i), \tag{10}$$

where $\mathcal{P}$ is performance of ROOT algorithm, $\mathcal{F}(i)$ is either $\mathcal{S}$ in (8) or $\mathcal{A}$ in (9). In [9], two definitions of ROOT in [8] i.e. survival time and average fitness were analyzed. Also, two different benchmark problems were proposed.

In [10], a new two-layer multi-objective method was proposed to find robust solutions that can maximize both survival time and average fitness. In [11], another multi-objective method was proposed to minimize switching cost and maximize survival time. A PSO algorithm was used as the optimizer. Additionally, the algorithm used the acceptance threshold for robust solutions similar to [8]. Euclidean distance between two solutions was used as the switching cost, and three different performance indicators were used:

$$\mathcal{F} = \frac{1}{T}\sum_{i=1}^{T} F_i, \tag{11}$$

$$\mathcal{R} = \frac{1}{T}\sum_{i=1}^{T} R_i, \tag{12}$$

$$\mathcal{C} = \frac{1}{T}\sum_{i=1}^{T} C_i, \tag{13}$$

where T is number of environments, F_i is fitness value of robust solutions, R_i is robustness (calculated by (8)) and C_i is switching cost in i^{th} environment. Switching cost is Euclidean distance between robust solutions in successive environments.

All of the proposed methods in [6], [8], [9] and [11] used predicted future fitness values of solutions for selecting robust solutions. In [6], an RBFN was used for approximating previous fitness values of solutions and an AR was used for predicting future values. In [8], the authors removed the approximation part and used true fitness values in previous environments for training the AR in order to remove the negative effect of approximation errors on the performance of the algorithms. In [9], the authors used the same methods as in [6] for approximation and prediction to investigate the performance of the proposed algorithms in [6] and [8]. In [11], the authors assumed that algorithms benefited from an ideal predictor without any error, so in their experiments the true future fitness values were used instead of the predicted values. However, removing the approximator and predictor from algorithms that work based on future fitness values of solutions clearly is a substantial simplification and the performance on a real-world problem where future fitness values are not available may be very different. Overall, for solving real-world problems, almost all the current ROOT methods [6], [8]–[10] and [11] need to use approximation and prediction methods based on time series [12].

B. Tracking Moving Optimum

Multi-population/multi-swarm methods are popular among researchers in the DOP domain [1], [13]. They consist of at least two sub-populations/swarms handling different tasks or separate regions in the problem space.

In [14], Self Organizing Scouts (SOS) was proposed which utilized a big sub-population for global search and a number of small sub-populations for tracking changes of identified peaks. This strategy has also been proposed with other meta-heuristics such as PSO [15]–[18] and artificial fish swarm optimization [19]–[21].

In [22], a speciation method was used to split the population into sub-populations. In [23], a regression-based approach (RSPSO) was presented to enhance the convergence rate using speciation-based methods. Every subpopulation was confined to a hypersphere around the best solution.

In [24], a method based on clustering was proposed for developing sub-populations, which was simplified and further improved in [25]. In [26], a method called MEPSO was proposed in which the population was divided into two parts. The first part was responsible for exploitation and the second one for exploration. Gaussian local search and differential mutation were used to improve diversity in the environment.

In [27], two multi-population methods, called MQSO and MCPSO, were proposed. In MQSO, quantum particles appear at random positions, uniformly distributed around the swarm's global best. In MCPSO, some or all of the particles in each swarm have a 'charge', and charged particles repel each other, leading to larger diversity. The population size is equal for every sub-swarm, and the number of sub-swarms is fixed and pre-determined. An anti-convergence method ensures continued search for possible better peaks. In addition, a mechanism called exclusion is used to avoid several swarms converging to the same peak.

A version of MQSO with an adaptive number of sub-populations, called AMQSO, was proposed in [28]. AMQSO starts with one sub-population and a new sub-population is created if all previous sub-populations have converged. This method has significantly improved the performance.

Li et al [29] proposed a method to adapt the number of populations based on statistical data on how many populations have found new peaks. If this number is large, more populations will be introduced and vice versa. Additionally, a new heuristic clustering, a population hibernation scheme, a population exclusion scheme, a peak hiding method and two movement methods (to track peaks and avoid stagnation) were proposed.

A PSO with two types of sub-swarms called finder-tracker multi-swarm PSO (FTmPSO) was proposed in [30]. The finder swarm finds new uncovered peaks. When it converges to a peak, it creates a new tracker swarm to track the peak.

An exclusion mechanism re-initializes the finder swarm if it converges to a peak that already has a tracker swarm on it. In addition, a mechanism to schedule tracker swarms called sleeping-awakening was proposed. It allocates more computational power to more promising swarms. Furthermore, a new method for re-diversification of tracker swarms (after a change) was proposed. The method re-initializes all particles randomly around Gbest [7] and their velocity vector is randomly set based on the peak's shift severity. In [31], a hybrid method based on FTmPSO was proposed for DOPs with previous-solution displacement restriction (PSDR).

III. The Proposed Framework

The literature review above shows that almost all current methods on ROOT need to use approximation and prediction methods based on time series [12]. The accuracy of this approach depends on the amount of data available, i.e. past and current fitness values covering the representative regions of the search space. In problems with a large number of dimensions and/or large search space and/or high change frequency, a very large amount of data is required to obtain an accurate approximation. This may be impossible to achieve. In this section, we propose a new framework for ROOT that does not rely on predicted future values of solutions. Consequently, the proposed framework does not require complicated approximation and prediction methods for predicting solution fitness values. Instead, a multi-population algorithm is responsible to find peaks, track them after environmental changes and gather information about their behavior. This information will be used to predict the future behavior of peaks. When the current solution becomes unacceptable, the next robust solution will be selected by a decision rule based on information collected by sub-populations such as shift severity or height severity. In this paper, we propose four such decision rules.

A. *The multi-population/multi-swarm method*

In this section, we describe the necessary characteristics of multi-population (or multi-swarm) methods that can be used inside the proposed ROOT framework. We assume a multi-population algorithm would continuously try to identify new peaks and track them after an environmental change. Knowledge about the problem such as number of peaks and their shift severities should not be necessary. Additionally, the algorithm should be able to adapt the number of populations as needed. For example, the proposed multi-population methods in [28], [30] have such characteristics.

The other requirement is preventing overcrowding, i.e., each peak should be covered by at most one sub-population. Typically, algorithms use an exclusion mechanism [27] for this purpose. If the distance between the best found positions of two populations drops below some exclusion radius r_{excl}, the population with the worse best found position is re-initialized. A good formula for calculating r_{excl} without a need to know the number of peaks was proposed in [28] as follows:

$$r_{\text{excl}} = excl_{\text{factor}} \times \frac{SR}{TSN^{\frac{1}{D}}}, \qquad (14)$$

where $excl_{\text{factor}}$ is a positive constant less than 1, SR is the search domain, TSN is the current number of sub-populations and D is the number of dimensions. It is worth mentioning that the original formula in [27] used number of peaks instead of TSN which usually is unknown in real-world problems. This was changed to TSN in [28]. Note that, in the proposed framework, each sub-population separately records some information about its covered peak. Therefore, the exclusion mechanism for the proposed framework should allow such a record to be transferred from one population to another before the population is re-initialized. If the surviving population is younger (according to the environment number that it was created), then before the algorithm re-initializes the older one, its database will be transferred to the surviving one.

Another characteristic that a compatible multi-population method should have is being able to track peaks. Therefore, the populations that are responsible to cover and track peaks need to be able to deal with diversity loss [1]. In [1], methods that deal with diversity were grouped into two categories: methods that maintain diversity during the search and the methods that introduce diversity when changes occur. Additionally, to track peaks, populations need to deal with the outdated memory issue that happens after environmental changes. In fact, after changes, the stored fitness values by the algorithm may have changed. This issue can be addressed by re-evaluating all individuals after environmental changes.

Finally, since the algorithms need to be able to react to an environmental change, e.g. by updating memory and calculating and storing some information such as shift severity of peaks, they need to know when a change has occurred. Since detecting a change is a separate issue and in many real-world dynamic environments the occurrence of a change is obvious (e.g., arrival of new order, change in temperature) [32], in this paper, as in all previous algorithms of ROOT [6], [8]–[11], we assume the information about environmental change events is known and does not need to be detected.

B. *New decision maker process for choosing robust solutions*

The proposed framework acts based on information gathered by sub-populations tracking peaks. Note that at the t^{th} environment, only sub-populations which were created at the $(t-2)^{\text{th}}$ environment and before that are considered. There are three types of information stored in each sub-population's database:

1. The Euclidean distance between best found positions (such as $Gbest$ in PSO) at the end of each successive pair of environments. The average of these distances indicates peaks Shift Severity.

$$S_i = \frac{1}{t-b_i-1} \times \sum_{k=b_i+1}^{t-1} \left\| \mathbf{g}_i^{(k),\text{end}} - \mathbf{g}_i^{(k-1),\text{end}} \right\|, \qquad (15)$$

where S_i is the estimated shift severity of the peak covered by the i^{th} sub-population, b_i is the environment in which the i^{th} sub-population has been created, t is the current environment number, and $\mathbf{g}_i^{(k),\text{end}}$ is the best found position of the i^{th} sub-population at the end of the k^{th} environment.

2. The differences between fitness values of its best found positions before and after each environmental change. The average of these values indicates the variance of fitness values of the best found position after environmental changes.

$$FV_i = \frac{1}{t - b_i} \times \sum_{k=b_i}^{t-1} \left| f\left(\mathbf{g}_i^{(k),\text{end}}, \theta^{(k)}\right) - f\left(\mathbf{g}_i^{(k+1),\text{beginning}}, \theta^{(k+1)}\right) \right|, \tag{16}$$

where FV_i is the fitness variance of the peak covered by the i^{th} sub-population, $f(\mathbf{g}_i^{(k),\text{end}}, \theta^{(k)})$ is the fitness value of the best found position by the i^{th} sub-population at the end of the k^{th} environment and $f(\mathbf{g}_i^{(k+1),\text{beginning}}, \theta^{(k+1)})$ is the re-evaluated fitness value of this position at the beginning of the next environment.

3. The fitness difference between best found positions at the end of each successive pair of environments. The average of these (called height variance) indicates a peak's height variability.

$$HV_i = \frac{1}{t - b_i - 1} \times \sum_{k=b_i+1}^{t-1} \left| f\left(\mathbf{g}_i^{(k),\text{end}}, \theta^{(k)}\right) - f\left(\mathbf{g}_i^{(k-1),\text{end}}, \theta^{(k-1)}\right) \right|, \tag{17}$$

where HV_i is the calculated height variance of the peak covered by the i^{th} sub-population. The database of each sub-population will be updated after each environmental change.

If at t^{th} environment, the fitness value of the current robust solution $\mathbf{r}$ is greater than the threshold V, then it will be kept for at least another environment. Otherwise, after the computational budget [9] η which is usually until the end of the current environment, the following procedure will be executed:

Step 1: Pre-selection: Remove from consideration each sub-population i if the current $f(\mathbf{g}_i, \theta^{(t)}) < (FV_i + V)$. FV_i shows how much the fitness value of a position on peak i (covered by i^{th} sub-population) is expected to change after an environmental change. Thus, if $f(\mathbf{g}_i, \theta^{(t)}) < (FV_i + V)$, in the next environment $f(\mathbf{g}_i, \theta^{(t+1)})$ will likely be below the threshold so this position is not considered a robust solution.

For the remaining candidates $\mathbf{g}$, the proposed framework executes the second step for choosing one of the candidates' $\mathbf{g}$ as the next robust solution. If there is no candidate peak, then the algorithm chooses the $\mathbf{g}$ with the highest fitness value.

Step 2: Four different strategies for choosing the next robust solution (NRS) are proposed as follows:

- The $\mathbf{g}$ with the highest fitness value minus its FV is chosen.

$$\text{NRS} = \text{argmax}_{i=1}^{e} \left(f(\mathbf{g}_i, \theta^{(t)}) - FV_i \right), \tag{18}$$

  where e is the number of candidate $\mathbf{g}$ remaining from the first step.
- The $\mathbf{g}$ with the lowest calculated shift severity S (15) is chosen.

$$\text{NRS} = \text{argmin}_{i=1}^{e}(S_i), \tag{19}$$

- The $\mathbf{g}$ with the lowest height variance calculated by (17) is chosen.

$$\text{NRS} = \text{argmin}_{i=1}^{e}(HV_i), \tag{20}$$

Algorithm 1: ROOT framework equipped with a multi-population method

```
1  Initialize multi-population method;
2  repeat
3      if an environmental change is happened then
4          forall sub-population do
5              Update Database;
6              Calculate $S$, $FV$ and $HV$ by (15), (16) and (17);
7              Update Memory;
8              Other actions for the embedded multi-population
                 method based on its procedure (such as
                 introducing diversity);
9      if computational budget $\eta$ is finished then
10         if the robust solution is not acceptable then
11             Identify candidate $\mathbf{g}$ by Step 1 in Section III-B;
12             Choose one of the candidates $\mathbf{g}$ based on a
                 strategy in Section III-B;
13     Execute an iteration of the multi-population method
         including finding and tracking peaks;
14     Create or remove sub-populations if needed (based on the
         procedure of the multi-population method);
15     forall pair of sub-populations $i$ and $j$ do
16         if $\|\mathbf{g}_i - \mathbf{g}_j\| < r_{\text{excl}}$ then
17             if $f(\mathbf{g}, \theta^{(t)})$ value of the younger one is better
                 then
18                 Copy the older ones database to the newer one;
19             Keep the sub-population with better $f(\mathbf{g}, \theta^{(t)})$ and
                 remove or the other one;
20     Update $r_{\text{excl}}$ by (14);
21 until stopping criterion is met;
```

- The $\mathbf{g}$ with the lowest value obtained by (21) is chosen.

$$\text{NRS} = \text{argmin}_{i=1}^{e} \left(\frac{S_i}{S_{\max}} + \frac{HV_i}{HV_{\max}} \right), \tag{21}$$

In the 4th strategy, both height variance HV and shift severity S are used. The values of HV and S of each candidate peak are divided by their maximum values ($S_{\max}$ and $HV_{\max}$) to be normalized in the range $(0, 1)$. The proposed framework checks the acceptability of the current robust solution. If it is not acceptable, it will execute steps 1 and 2 above to choose the next robust solution. If there is no option, the best $\mathbf{g}$ is chosen as NRS. The pseudo code of the proposed framework is shown in Algorithm 1.

IV. Experiments and Analysis

A. Performance indicators

We focus on the most important goal of ROOT, survival time. We will use the performance indicator in (10) for the survival time definition in (8). Furthermore, the performance indicator in (11) is used to show the average fitness value of robust solutions when we compare our methods with the state-of-the-art ROOT algorithms.

B. Benchmark functions

Moving peaks benchmark (MPB) [33] is the most popular benchmark function in the DOP field. In its standard form, all peaks are behaving identical, so no solution is more robust than another. This is why in ROOT researchers used various modified versions [6], [8]–[11], [34].

In [6] the authors used three different benchmark generators, namely the modified MPB with different height and width severities for each peak; the modified dynamic rotation generator [35] with different height and width severities for each peak; and finally the modified dynamic composition benchmark generator [35] with only different height severity for each peak. Each of these three benchmark generators was used with three different numbers of dimensions which resulted in nine test instances in total. In [8] and [34], authors used a modified version of MPB with different height and width severities. One problem instance of this version was used for testing the algorithm on a 2-dimensional search space. In [9], two different benchmark problems were proposed, one specifically designed for maximizing survival time and another for maximizing average fitness. These two benchmarks used two different modified versions of the baseline fitness function of MPB. Furthermore, rotation rather than translation was used to move peaks after environmental changes. The authors used six different dynamics [35] on their two benchmarks. In [11], authors used a modified MPB with different height and width severities for peaks. For changing heights and widths of peaks, the benchmark used three different dynamics: small step, random and recurrent [35], but they used the standard peak center relocation also used in the standard MPB [33].

In this paper, and similar to ROOT papers in [6], [8], [10], [11], [34], we use the standard baseline function of MPB as follows:

$$f^{(t)}(\mathbf{x}) = \max_{i=1}^{m} \left\{ h_i^{(t)} - \left(w_i^{(t)} \cdot \left\| \mathbf{x} - \mathbf{c}_i^{(t)} \right\| \right) \right\}, \tag{22}$$

where m is the number of peaks, $\mathbf{x}$ is a solution in the problem space, $h_i^{(t)}$, $w_i^{(t)}$ and $\mathbf{c}_i^{(t)}$ are the height, width and center of the i^{th} peak in the t^{th} environment, respectively. In the modified version of MPB for ROOT (mMPBR) used in this paper, each peak has its own height and width severity. This is similar to the benchmarks in previous ROOT papers [6], [8], [10], [11], [34]. Additionally, we use different shift severities for different peaks, although in the experiments we also investigate the effect of having the same shift severity for all peaks. The reason for having different height, width and shift severities for each peak is to have different level of robustness among them. The height, width and center of a peak change from one environment to the next as follows:

$$h_i^{(t+1)} = h_i^{(t)} + \alpha_i \cdot \mathcal{N}(0,1), \tag{23}$$

$$w_i^{(t+1)} = w_i^{(t)} + \beta_i \cdot \mathcal{N}(0,1), \tag{24}$$

$$\mathbf{c}_i^{(t+1)} = \mathbf{c}_i^{(t)} + \mathbf{v}_i^{(t+1)}, \tag{25}$$

where

$$\mathbf{v}_i^{(t+1)} = s_i \cdot \frac{(1-\lambda)\cdot\mathcal{R} + \lambda\cdot\mathbf{v}_i^{(t)}}{\left\|(1-\lambda)\cdot\mathcal{R} + \lambda\cdot\mathbf{v}_i^{(t)}\right\|}, \tag{26}$$

where $\mathcal{N}(0,1)$ represents a random number drawn from a Gaussian distribution with mean 0 and variance 1, α_i is the height severity, β_i is the width severity, s_i is the shift severity of the i^{th} peak, $\mathcal{R}$ is a uniformly generated random vector $\in [-0.5, 0.5]$ and λ is the correlation coefficient.

The parameter settings of the mMPBR are shown in Table I. The highlighted values in Table I are default parameter values of mMPBR which build the default scenario of the benchmark in this paper. In the experiments, different number of peaks, change frequencies, dimensions and shift severities are used in order to test the sensitivity of the proposed algorithm. For investigating the impact of different parameter settings of mMPBR on the algorithms' performance, we keep most of the default parameter settings and change 1 or 2 parameters to build each experiment.

TABLE I
PARAMETER SETTINGS OF MMPBR (DEFAULT VALUES ARE HIGHLIGHTED)

Parameter	Value(s)
Number of peaks, m	2,5,10,20,30,50,100,200
Evaluations between changes, f	1000,2500,5000
Shift severity, s	1,5,randomized in [0.5,1],[0.5,3],[0.5,5]
Height severity, α	Randomized in [1,15]
Width severity, β	Randomized in [0.1,1.5]
Peaks shape	Cone
Correlation coefficient, λ	0
Number of dimensions, D	2,**5**,10
Peak location range, SR	[-50,50]
Peak height range	[30,70]
Peak width range	[1,12]
Initial height value	50
Initial width value	6
Number of environments	100

C. Algorithms and parameter settings

In the experiments, we use FTmPSO [30] inside the proposed framework as the multi-swarm method. There are three major reasons for this choice. First, it is very simple, which makes it easy to analyze the impact of the framework on performance. Second, it is a competitive TMO algorithm. Third, with minimal modifications, this method is compatible with the framework according to Section III-A: (a) it uses (14) for determining the exclusion radius r_{excl}; (b) its exclusion mechanism allows the transfer of peak information from one swarm to another; (c) it uses the learned shift severity (15) instead of the true shift that was used in the original paper. Additionally, we do not use the exploiter particle and awakening-sleeping mechanisms proposed in its original paper. The reason is that these two mechanisms improve the exploitation on the best peak which is not useful in ROOT. Readers are referred to [30] for more details of this multi-swarm algorithm. Integrated in the framework, the algorithm has four versions depending on the chosen strategies (Section III-B). The four versions are RFTmPSO-s1 to RFTmPSO-s4, based on strategies 1 to 4, respectively.

The parameter setting of FTmPSO inside the proposed framework is shown in Table II. Since the task of the multi-population methods in the proposed framework is similar to their original purpose (TMO), parameter settings suggested in the original paper can be used here as well. A sensitivity analysis on RFTmPSO is provided in supplementary materials to illustrate the effect of different FTmPSO parameter settings on the ROOT performance. Based on this analysis, the parameter settings in Table II have been chosen.

TABLE II
PARAMETER SETTINGS OF FTMPSO

Parameter	Value(s)
C_1, C_2	2.05
χ	0.729843788
Tracker-swarm's Population Size	5
$excl_{\text{factor}}$	0.1
r_{excl}	calculated by (14)
Finder-swarm's Population Size	10
P	1
Q	1
$Conv - limit$	1
k	10
Stop criterion	Max fitness evaluation number

V. EXPERIMENTAL RESULTS

We report experimental results in two parts. In the first part, we investigate the performance of the proposed framework with four strategies from Section III-B on several problem instances with different characteristics. The second part compares our proposed methods embedded into different multi-swarm methods with the state-of-the-art ROOT methods and compares their behaviors on different problem instances.

All experimental results are obtained by performing 30 independent runs. To test the statistical significance of the reported results, we perform a multiple comparison test and the best results based on Wilcoxon signed-rank test with Holm-Bonferroni p-value correction and $\alpha = 0.05$ are highlighted in each table. If there are more than one highlighted results, it means they are not significantly different.

A. Analyzing the proposed framework on problems with different characteristics

Table III shows the average survival time of RFTmPSO with four different strategies on mMPBR with different numbers of peaks. All other mMPBR parameters are set to default values. The worst results are obtained in mMPBR with 2 peaks. All versions of RFTmPSO perform identical on this instance because the number of options for choosing the next robust solution is limited. Also, when the number of peaks is low, there are large areas of low fitness because there are few peaks to cover these areas. As a result, the average solution quality is lower, and robust solutions can lose their quality more quickly. By increasing the number of peaks, the average survival time increases because peaks are likely to overlap and support robust solutions. Increasing the number of peaks also increases the performance difference between different versions of RFTmPSO because there are more peaks with different characteristics and RFTmPSO has more options to choose the best of them based on the robust solution selection strategies. The best results are obtained on mMPBR with 50 peaks, but when the number of peaks is increased to 100 and 200, performance decreases. The reason is that the algorithm can no longer cover all peaks because of their large number. Furthermore, the algorithm cannot perform a good local search to track peaks because the number of tracker swarms is large.

In problems with a higher number of peaks such as 100 and 200, the density of peaks is high. As a result, it is highly likely that some peaks are covered by higher peaks. In such case, the tracker swarm will lose its covered peak, and hence their associated information, leading to a worse performance. However, the multi-population algorithm would search for possible uncovered peaks all the time (Section III-A). Therefore, when a peak hidden by another peak re-appears, the multi-swarm algorithm would be able to find it and start gathering information about it again. Although algorithm performances are worse for 100 and 200 peaks in comparison with 50 peaks, the average survival time values are still very good. This demonstrates the ability of the proposed methods in dealing with a higher number of peaks.

When increasing the threshold V, the performance of RFTmPSO decreases because the survival time of solutions in the problem space decreases. No algorithm can do anything about this. Also, the performance of RFTmPSO versions are closer when V is high because the number of options for choosing the next robust solution decreases.

Table III shows that RFTmPSO-s2 performs better than RFTmPSO-s3. This illustrates that the effect of shift severity on the life cycle length of robust solutions is more important than the effect of height variance. However, when we consider both parameters (RFTmPSO-s4), as in (21), the performance is improved. RFTmPSO-s4 performs best overall. RFTmPSO-s1 could rarely outperform other versions of RFTmPSO which means that considering fitness variance in (18) for choosing the next robust solution is not the best way.

Table IV shows the obtained average survival time for RFTmPSO for mMPBR with different numbers of peaks, different numbers of dimensions and default values for other parameters. The proposed RFTmPSO algorithms can find robust solutions in high numbers of dimensions and peaks. When the peak number increases to 50, the performance improves regardless of the number of dimensions. Increasing the number of peaks further to 100 or 200 leads to a slight deterioration of results. The average survival time is also lower because the problems become more complex for algorithms. Overall, RFTmPSO-s4 maintains its superiority.

Table V shows the results of testing RFTmPSO on mMPBR with different shift severities in 5 and 10 dimensions, with default values for other parameters. As expected, when shift severity increases, the average survival time decreases because tracking peaks with higher shift severities is harder for tracker swarms and their ability of gathering information decreases. More importantly, the maximal possible survival time decreases due to the increased shift severities. Also, robust solutions become unacceptable more quickly because peaks move with larger steps. The worst results are observed when all peaks have the same high shift severity of 5. When all peaks have the same severity, information on a peaks shift severity is not useful. Thus, RFTmPSO-s4 does not perform better than other algorithms because it relies on learning the difference of peak shift severities. On the other hand, RFTmPSO-s3, which does not use information about shift severity, has the

TABLE III
AVERAGE SURVIVAL TIME (AND STANDARD ERROR) ON mMPBR WITH DIFFERENT PEAK NUMBER m, $f = 2500$, s RANDOMIZED $\in [0.5, 3]$ AND $D = 5$.

V	Algorithm	Peak Number, m							
		2	5	10	20	30	50	100	200
40	RFTmPSO-s1	3.08(0.65)	3.98(0.39)	4.80(0.62)	5.48(0.60)	6.46(0.75)	7.41(0.83)	6.19(0.47)	6.26(0.55)
	RFTmPSO-s2	3.41(0.80)	4.13(0.46)	4.53(0.67)	5.60(0.82)	8.11(1.18)	7.84(0.99)	5.73(0.47)	6.11(0.60)
	RFTmPSO-s3	3.36(0.80)	3.81(0.45)	4.21(0.61)	4.89(0.81)	6.65(1.08)	7.01(0.96)	5.87(0.71)	5.80(0.61)
	RFTmPSO-s4	3.36(0.81)	3.86(0.45)	4.67(0.68)	6.14(0.85)	8.21(1.16)	8.23(0.98)	6.51(0.46)	6.89(0.62)
40	RFTmPSO-s1	2.19(0.52)	2.55(0.31)	3.34(0.40)	3.90(0.34)	4.31(0.39)	5.20(0.47)	4.98(0.60)	4.90(0.43)
	RFTmPSO-s2	2.30(0.58)	2.59(0.30)	3.16(0.38)	3.63(0.33)	5.23(0.73)	5.73(0.63)	5.05(0.55)	5.07(0.34)
	RFTmPSO-s3	2.29(0.58)	2.40(0.29)	3.00(0.37)	3.44(0.34)	4.77(0.57)	5.95(0.76)	4.36(0.55)	4.50(0.39)
	RFTmPSO-s4	2.30(0.55)	2.48(0.28)	3.23(0.38)	4.22(0.41)	5.31(0.61)	6.16(0.62)	5.43(0.58)	5.44(0.37)
40	RFTmPSO-s1	1.33(0.39)	1.51(0.19)	2.40(0.35)	2.55(0.21)	3.26(0.46)	3.65(0.40)	3.17(0.25)	3.31(0. 33)
	RFTmPSO-s2	1.35(0.39)	1.51(0.18)	2.10(0.25)	2.43(0.26)	3.19(0.34)	3.94(0.46)	3.27(0.40)	3.33(0.30)
	RFTmPSO-s3	1.34(0.39)	1.46(0.17)	2.02(0.24)	2.51(0.28)	2.90(0.32)	3.93(0.55)	3.22(0.31)	3.20(0.32)
	RFTmPSO-s4	1.34(0.39)	1.51(0.18)	2.18(0.27)	2.77(0.31)	3.67(0.56)	4.10(0.51)	3.39(0.37)	3.57(0.29)

TABLE IV
AVERAGE SURVIVAL TIME (AND STANDARD ERROR) ON mMPBR WITH DIFFERENT m, DIFFERENT D, $f = 2500$ AND s RANDOMIZED IN [0.5,3].

V	Algorithm	$m = 5$			$m = 10$			$m = 20$			$m = 50$			$m = 100$		
		D=2	D=5	D=10	D=2	D=5	D=10	D=2	D=5	D=10	D=2	D=5	D=10	D=2	D=5	D=10
40	RFTmPSO-s1	4.83 (0.87)	3.98 (0.39)	3.91 (0.75)	6.14 (0.92)	4.80 (0.62)	4.33 (0.47)	7.42 (1.09)	5.48 (0.60)	4.35 (0.35)	7.87 (1.04)	7.41 (0.83)	5.37 (0.44)	7.41 (2.00)	6.19 (0.47)	4.97 (0.43)
	RFTmPSO-s2	4.98 (0.86)	4.13 (0.46)	3.77 (0.89)	7.23 (1.59)	4.53 (0.67)	4.39 (0.52)	9.26 (1.32)	5.60 (0.82)	4.22 (0.34)	7.91 (0.68)	7.84 (0.99)	5.60 (0.50)	7.50 (1.44)	5.73 (0.47)	4.62 (0.44)
	RFTmPSO-s3	4.08 (0.44)	3.81 (0.45)	3.76 (0.88)	6.82 (1.41)	4.21 (0.61)	4.42 (0.57)	7.48 (1.17)	4.89 (0.81)	4.05 (0.43)	8.15 (0.98)	7.01 (0.96)	5.46 (0.78)	6.91 (0.69)	5.87 (0.71)	5.21 (0.46)
	RFTmPSO-s4	4.84 (0.86)	3.86 (0.45)	3.83 (0.88)	8.03 (1.62)	4.67 (0.68)	4.49 (0.56)	9.71 (1.40)	6.14 (0.85)	4.22 (0.38)	8.47 (1.04)	8.23 (0.98)	5.93 (0.69)	8.29 (1.34)	6.51 (0.46)	5.30 (0.39)
45	RFTmPSO-s1	3.15 (0.48)	2.55 (0.31)	2.45 (0.43)	4.66 (0.87)	3.34 (0.40)	3.23 (0.41)	5.11 (0.63)	3.90 (0.34)	3.29 (0.29)	6.52 (1.02)	5.20 (0.47)	4.50 (0.48)	6.42 (0.58)	4.98 (0.60)	3.29 (0.24)
	RFTmPSO-s2	3.24 (0.49)	2.59 (0.30)	2.46 (0.43)	5.24 (1.14)	3.16 (0.38)	2.95 (0.34)	5.67 (0.68)	3.63 (0.33)	3.30 (0.29)	6.10 (0.59)	5.73 (0.63)	4.74 (0.58)	6.06 (0.48)	5.05 (0.55)	3.26 (0.25)
	RFTmPSO-s3	3.31 (0.53)	2.40 (0.29)	2.36 (0.42)	5.17 (1.15)	3.00 (0.37)	3.02 (0.38)	5.03 (0.64)	3.44 (0.34)	3.25 (0.33)	6.27 (0.62)	5.95 (0.76)	4.81 (0.57)	5.72 (0.55)	4.36 (0.55)	3.68 (0.28)
	RFTmPSO-s4	3.30 (0.51)	2.48 (0.28)	2.46 (0.42)	5.32 (1.14)	3.23 (0.38)	3.00 (0.38)	5.65 (0.68)	4.22 (0.41)	3.51 (0.33)	7.03 (1.23)	6.16 (0.62)	5.03 (0.57)	6.48 (0.52)	5.43 (0.58)	3.87 (0.27)
50	RFTmPSO-s1	1.99 (0.36)	1.51 (0.19)	1.48 (0.33)	2.40 (0.37)	2.40 (0.35)	1.85 (0.26)	3.32 (0.39)	2.55 (0.21)	2.26 (0.21)	4.52 (0.86)	3.65 (0.40)	2.99 (0.35)	4.27 (0.36)	3.17 (0.25)	2.34 (0.14)
	RFTmPSO-s2	1.92 (0.30)	1.51 (0.18)	1.45 (0.32)	2.57 (0.40)	2.10 (0.25)	1.94 (0.28)	3.58 (0.45)	2.43 (0.26)	2.11 (0.21)	4.74 (0.83)	3.94 (0.46)	3.18 (0.36)	4.49 (0.40)	3.27 (0.40)	2.25 (0.16)
	RFTmPSO-s3	1.87 (0.31)	1.46 (0.17)	1.48 (0.32)	2.46 (0.40)	2.02 (0.24)	1.94 (0.28)	3.49 (0.46)	2.51 (0.28)	2.23 (0.25)	4.62 (0.65)	3.93 (0.55)	3.06 (0.36)	3.75 (0.37)	3.22 (0.31)	2.51 (0.15)
	RFTmPSO-s4	1.95 (0.33)	1.51 (0.18)	1.48 (0.33)	2.68 (0.40)	2.18 (0.27)	1.97 (0.29)	3.75 (0.45)	2.77 (0.31)	2.19 (0.23)	4.19 (0.54)	4.10 (0.51)	3.18 (0.34)	4.57 (0.39)	3.39 (0.37)	2.64 (0.17)

best results on these problems.

On instances in which each peak has its own randomly generated shift severity, RFTmPSO-s4 and RFTmPSO-s2 obtain the best results. In these instances, some peaks have higher values of shift severity which make them less reliable for carrying robust solutions and vice versa. Therefore, algorithms that learn about shift severities such as RFTmPSO-s4 and RFTmPSO-s2 can find more robust solutions. RFTmPSO-s4 obtains the best results due to using both types of information (shift severity and HeightVar). Similar to Table IV, in Table V the average survival time values are lower in 10-dimensions than in 5-dimensions.

Table VI shows the average survival time by RFTmPSO in mMPBR with different numbers of peaks and change frequencies, with default values for other parameters. Like in previous experiments, RFTmPSO-s4 has better performance overall in environments with higher change frequencies. In problem instances with fewer evaluations per change (lower f, higher change frequency), the average survival time decreases because the accuracy of gathered information and the local search in each peak decrease. This is due to a lack of time to react to changes. For $f = 500$, the difference between methods is small due to lower information accuracy. When f increases, the difference between the methods becomes more noticeable.

The average survival time in problems with a small number of peaks does not decrease significantly when f is small. The reason is that a small number of peaks means a small number of sub-swarms, so the algorithm has enough time for exploitation before the next environmental change. On the other hand, when the number of peaks is high, the algorithm has many sub-swarms and so can perform fewer iterations of exploiting before the next environmental change. This leads to less accurate information and lower performance.

TABLE V
AVERAGE SURVIVAL TIME (AND STANDARD ERROR) ON MMPBR WITH DIFFERENT SHIFT SEVERITIES s, $D = 5$, $m = 10, 20$ AND $f = 2500$.

V	Algorithm	5 Dimensional					10 Dimensional				
		s=1	s=5	s=r(0.5,1)	s=r(0.5,3)	s=r(0.5,5)	s=1	s=5	s=r(0.5,1)	s=r(0.5,3)	s=r(0.5,5)
40	RFTmPSO-s1	7.87(0.80)	1.21(0.10)	10.17(0.64)	5.48(0.60)	5.35(0.58)	5.26(0.37)	1.05(0.07)	9.19(1.11)	4.35(0.35)	3.94(0.46)
	RFTmPSO-s2	7.47(0.61)	1.08(0.07)	11.64(1.17)	5.60(0.82)	5.98(0.88)	4.02(0.32)	1.02(0.08)	8.74(1.12)	4.22(0.34)	3.75(0.41)
	RFTmPSO-s3	8.26(0.69)	1.20(0.12)	10.40(0.92)	4.89(0.81)	5.45(0.91)	5.68(0.57)	1.08(0.07)	9.63(1.26)	4.05(0.43)	3.70(0.50)
	RFTmPSO-s4	8.10(0.81)	1.16(0.10)	11.96(1.14)	6.14(0.85)	5.94(1.00)	5.34(0.39)	1.08(0.07)	9.77(1.13)	4.22(0.38)	3.91(0.44)
40	RFTmPSO-s1	5.98(0.56)	0.78(0.07)	7.63(0.64)	3.90(0.34)	3.59(0.41)	4.02(0.34)	0.73(0.05)	6.65(0.95)	3.29(0.29)	2.68(0.28)
	RFTmPSO-s2	5.76(0.56)	0.73(0.05)	6.87(0.60)	3.63(0.33)	3.51(0.46)	3.25(0.28)	0.70(0.06)	6.61(1.12)	3.30(0.29)	2.67(0.28)
	RFTmPSO-s3	6.50(0.68)	0.79(0.06)	8.02(0.73)	3.44(0.34)	3.67(0.56)	4.17(0.39)	0.75(0.06)	7.22(1.03)	3.25(0.33)	2.65(0.31)
	RFTmPSO-s4	6.42(0.71)	0.77(0.06)	8.38(0.72)	4.22(0.41)	3.85(0.50)	4.10(0.34)	0.74(0.06)	7.62(1.16)	3.51(0.33)	2.79(0.32)
40	RFTmPSO-s1	4.49(0.52)	0.43(0.05)	5.21(0.37)	2.55(0.21)	2.18(0.26)	2.56(0.22)	0.40(0.03)	4.86(1.00)	2.26(0.21)	1.80(0.25)
	RFTmPSO-s2	3.82(0.37)	0.41(0.04)	4.72(0.51)	2.43(0.26)	2.43(0.36)	1.99(0.14)	0.40(0.03)	4.75(0.98)	2.11(0.21)	1.80(0.21)
	RFTmPSO-s3	4.63(0.50)	0.45(0.04)	5.50(0.60)	2.51(0.28)	2.40(0.36)	2.63(0.22)	0.41(0.03)	5.36(1.09)	2.23(0.25)	1.74(0.27)
	RFTmPSO-s4	4.32(0.45)	0.42(0.04)	5.61(0.62)	2.77(0.31)	2.45(0.35)	2.62(0.21)	0.41(0.03)	5.43(1.09)	2.19(0.23)	1.81(0.27)

TABLE VI
AVERAGE FITNESS VALUE (AND STANDARD ERROR) ON MMPBR WITH DIFFERENT m AND EVALUATION BETWEEN CHANGES f, s RANDOMIZED IN [0.5,3] AND D=5.

V	Algorithm	$m = 5$			$m = 10$			$m = 20$			$m = 50$			$m = 100$		
		f=500	f=1000	f=2500	f=500	f=1000	f=2500	f=500	f=1000	f=2500	f=500	f=1000	f=2500	f=500	f=1000	f=2500
40	RFTmPSO-s1	2.80 (0.28)	3.54 (0.73)	3.98 (0.39)	4.03 (0.34)	4.13 (0.46)	4.80 (0.62)	4.37 (0.28)	4.89 (0.38)	5.48 (0.60)	5.35 (0.38)	5.61 (0.65)	7.41 (0.83)	5.00 (0.43)	5.27 (0.60)	6.19 (0.47)
	RFTmPSO-s2	2.49 (0.29)	3.78 (0.79)	4.13 (0.46)	3.90 (0.32)	4.25 (0.49)	4.53 (0.67)	4.36 (0.30)	5.10 (0.40)	5.60 (0.82)	5.55 (0.39)	5.86 (0.70)	7.84 (0.99)	4.95 (0.51)	5.52 (0.68)	5.73 (0.47)
	RFTmPSO-s3	2.77 (0.29)	3.32 (0.65)	3.81 (0.45)	3.92 (0.42)	4.03 (0.44)	4.21 (0.61)	4.25 (0.35)	4.44 (0.33)	4.89 (0.81)	5.46 (0.40)	5.63 (0.57)	7.01 (0.96)	4.98 (0.49)	4.75 (0.71)	5.87 (0.71)
	RFTmPSO-s4	2.80 (0.28)	3.90 (0.80)	3.86 (0.45)	4.01 (0.29)	4.37 (0.48)	4.67 (0.68)	4.42 (0.27)	5.00 (0.41)	6.14 (0.85)	5.50 (0.41)	5.94 (0.70)	8.23 (0.98)	5.36 (0.56)	5.63 (0.70)	6.51 (0.46)
45	RFTmPSO-s1	2.04 (0.23)	2.06 (0.44)	2.55 (0.31)	2.80 (0.29)	3.05 (0.33)	3.34 (0.40)	3.28 (0.23)	3.51 (0.30)	3.90 (0.34)	3.87 (0.30)	4.27 (0.50)	5.20 (0.47)	3.60 (0.34)	3.68 (0.40)	4.98 (0.60)
	RFTmPSO-s2	1.94 (0.18)	2.23 (0.49)	2.59 (0.30)	2.82 (0.30)	3.04 (0.35)	3.16 (0.38)	3.27 (0.21)	3.48 (0.31)	3.63 (0.33)	3.94 (0.29)	4.08 (0.67)	5.73 (0.63)	3.71 (0.36)	3.81 (0.47)	5.05 (0.55)
	RFTmPSO-s3	2.05 (0.25)	2.13 (0.48)	2.40 (0.29)	2.77 (0.28)	3.08 (0.38)	3.00 (0.37)	3.23 (0.24)	3.35 (0.29)	3.44 (0.34)	3.92 (0.30)	4.33 (0.78)	5.95 (0.76)	3.59 (0.31)	4.07 (0.49)	4.36 (0.55)
	RFTmPSO-s4	2.07 (0.22)	2.23 (0.48)	2.48 (0.28)	2.85 (0.33)	3.11 (0.38)	3.23 (0.38)	3.33 (0.24)	3.90 (0.37)	4.22 (0.41)	3.98 (0.30)	4.41 (0.69)	6.16 (0.62)	3.84 (0.37)	4.00 (0.47)	5.43 (0.58)
50	RFTmPSO-s1	1.14 (0.10)	1.37 (0.24)	1.51 (0.19)	1.87 (0.22)	1.95 (0.29)	2.40 (0.35)	2.04 (0.16)	2.30 (0.23)	2.55 (0.21)	2.42 (0.14)	2.88 (0.32)	3.65 (0.40)	2.31 (0.25)	2.48 (0.33)	3.17 (0.25)
	RFTmPSO-s2	1.15 (0.10)	1.42 (0.24)	1.51 (0.18)	1.86 (0.20)	2.03 (0.29)	2.10 (0.25)	2.04 (0.17)	2.33 (0.19)	2.43 (0.26)	2.42 (0.15)	2.57 (0.25)	3.94 (0.46)	2.40 (0.26)	2.37 (0.35)	3.27 (0.40)
	RFTmPSO-s3	1.13 (0.11)	1.33 (0.24)	1.46 (0.17)	1.83 (0.20)	1.96 (0.32)	2.02 (0.24)	2.02 (0.15)	2.31 (0.20)	2.51 (0.28)	2.42 (0.15)	3.15 (0.34)	3.93 (0.55)	2.29 (0.25)	2.56 (0.41)	3.22 (0.31)
	RFTmPSO-s4	1.17 (0.10)	1.48 (0.24)	1.51 (0.18)	1.93 (0.21)	2.10 (0.32)	2.18 (0.27)	2.09 (0.17)	2.44 (0.21)	2.77 (0.31)	2.44 (0.16)	3.00 (0.28)	4.10 (0.51)	2.44 (0.26)	2.55 (0.37)	3.39 (0.37)

B. Comparison with other methods

According to the reported results in Tables III to VI and based on the multiple comparison statistical analysis, the fourth strategy outperforms other strategies of the proposed framework. In this part, we use three different multi-swarm methods including FTmPSO [30], AmQSO [28] and mNAFSA [20] inside the proposed ROOT framework in combination with Strategy 4 (s4) to investigate the effect of the multi-swarm methods performance on the ROOT framework. These three algorithms are called RFTmPSO-s4, RAmQSO-s4 and RmNAFSA-s4, and are compared against three existing methods. The first method is a TMO algorithm based on FTmPSO [30] in which, when the current robust solution is not acceptable, the algorithm simply chooses the best found position as the next robust solution. Parameter settings of FTmPSO are the same as reported in Table II and parameter settings of AmQSO and mNAFSA are as proposed in their original references [20], [28]. As mentioned before, since the task of the multi-swarm methods in the proposed framework is the same as their original purpose, i.e, TMO, parameter settings suggested in the original papers can be used here as well. For RAmQSO-s4 and RmNAFSA-s4, we use the same exclusion mechanism as RFTmPSO-s4 with the same $excl_{\text{factor}}$ value. Additionally, both of them use the obtained value for shift severities in (15) instead of the actual value as an initial knowledge.

The other two methods are two reproduced versions of the method proposed in [8], which are considered the state-of-the-art in the field of ROOT at the moment [8]–[11]. The first version is exactly what was implemented in [8], utilizing the true values of previous environments instead of approximated values for training predictors, i.e., it assumes it does not need to use an approximator because it has access to the true values of previous environments. We will call it ROOT with predicted values (ROOT-PV) and the parameter settings of PSO and AR are the same as those used in [8]. The second version is reproduced from [11], in which the ROOT algorithm even

TABLE VII
AVERAGE SURVIVAL TIME AND FITNESS VALUES (AND STD. ERR) ON TEST INSTANCES WITH DIFFERENT DIMENSION D AND PEAK NUMBER m, F=2500 AND s RANDOMIZED IN [0.5,3]. BEST RESULTS BASED ON WILCOXON SIGNED-RANK TEST WITH HOLM-BONFERRONI p-VALUE CORRECTION, $\alpha = 0.05$ ARE HIGHLIGHTED, IGNORING ROOT-TFV DUE TO ITS UNREALISTIC ASSUMPTION OF KNOWING THE TRUE FUTURE FITNESS.

V	Algorithm	Survival time				Fitness value of robust solutions			
		D=2,P=5	D=2,P=20	D=5,P=5	D=5,P=20	D=2,P=5	D=2,P=20	D=5,P=5	D=5,P=20
40	ROOT-PV	3.10(0.39)	5.64(0.80)	0.83(0.16)	2.26(0.61)	48.35(0.32)	50.79(0.33)	-53.99(15.53)	20.49(3.04)
	ROOT-TFV	~~5.74(0.47)~~	~~8.06(0.89)~~	~~1.26(0.19)~~	~~3.29(0.33)~~	49.82(0.18)	51.64(0.21)	-54.37(13.45)	32.22(1.46)
	FTmPSO(TMO)	4.34(0.79)	6.18(1.00)	3.49(0.34)	4.22(0.33)	53.94(0.26)	55.53(0.20)	53.04(0.25)	54.68(0.16)
	RAmQSO-s4	5.40(0.91)	6.20(0.56)	4.15(0.87)	5.58(0.63)	52.19(0.32)	52.57(0.28)	51.45(0.32)	50.88(0.36)
	RmNAFSA-s4	4.72(0.83)	7.45(1.09)	3.71(0.52)	5.70(0.49)	52.60(0.35)	52.46(0.27)	51.48(0.29)	51.09(0.32)
	RFTmPSO-s4	4.84(0.86)	9.71(1.40)	3.86(0.45)	6.14(0.85)	52.52(0.36)	52.40(0.29)	51.02(0.31)	51.26(0.41)
45	ROOT-PV	2.71(0.26)	4.91(1.15)	0.13(0.05)	1.11(0.23)	50.70(0.39)	53.51(0.20)	-124.90(17.06)	2.92 (6.80)
	ROOT-TFV	~~3.93(0.36)~~	~~6.87(0.60)~~	~~0.29(0.08)~~	~~1.58(0.20)~~	51.22(0.24)	54.24(0.23)	-133.3(16.58)	16.19(3.92)
	FTmPSO(TMO)	2.97(0.57)	3.71(0.42)	2.22(0.26)	3.26(0.22)	56.47(0.18)	58.24(0.16)	56.02(0.21)	57.22(0.15)
	RAmQSO-s4	3.39(0.67)	4.79(0.56)	2.64(0.59)	4.16(0.55)	55.00(0.27)	55.93(0.23)	54.90(0.21)	54.23(0.27)
	RmNAFSA-s4	3.28(0.59)	4.96(0.60)	2.40(0.33)	4.15(0.38)	54.95(0.26)	55.52(0.25)	54.91(0.23)	54.44(0.29)
	RFTmPSO-s4	3.30(0.51)	5.65(0.68)	2.48(0.28)	4.22(0.41)	55.34(0.26)	55.14(0.23)	54.55(0.21)	54.53(0.26)
50	ROOT-PV	1.68(0.22)	2.83(0.37)	0.04(0.06)	0.37(0.23)	51.40(0.60)	56.45(0.12)	-190.50(19.76)	-37.01(7.55)
	ROOT-TFV	~~2.48(0.34)~~	~~4.36(0.26)~~	~~0.19(0.05)~~	~~0.63(0.11)~~	51.49(0.61)	56.66(0.09)	-116.73(14.48)	-11.98(6.18)
	FTmPSO(TMO)	1.79(0.31)	2.41(0.19)	1.29(0.16)	2.09(0.18)	58.79(0.19)	61.02(0.13)	58.56(0.23)	60.04(0.14)
	RAmQSO-s4	2.16(0.38)	3.21(0.50)	1.63(0.46)	2.55(0.28)	57.47(0.30)	59.15(0.13)	57.73(0.18)	57.94(0.16)
	RmNAFSA-s4	1.95(0.35)	3.44(0.48)	1.52(0.19)	2.51(0.22)	57.88(0.24)	58.47(0.17)	57.51(0.22)	57.81(0.19)
	RFTmPSO-s4	1.95(0.33)	3.75(0.45)	1.51(0.18)	2.77(0.31)	58.09(0.22)	58.36(0.19)	57.68(0.21)	58.14(0.18)

had access to the future true values instead of having to approximate past fitness functions and predict future values. We will call this version ROOT with true future values (ROOT-TFV). Note that ROOT-TFV is the ROOT method proposed in [8] using the true future values, and was used in [11].

The reason behind choosing ROOT-TFV in our comparisons is to investigate the effect of prediction error on the performance of the ROOT algorithm. For PSO in ROOT-TFV, we used the same parameter setting as ROOT-PV. We will not consider the obtained results of ROOT-PV in environments for which the training datasets are not complete. Note that ROOT-PV and even more so ROOT-TFV have access to information that is not available in real-world optimization, and thus results can only be taken as an upper bound of what these methods are able to achieve in practice. As mentioned before, we assume that the algorithms are informed when environmental changes happen.

The experiments in this section are done on four different test instances of mMPBR on 2 and 5 dimensions with 5 and 20 peaks (all other parameters have default values). This combination shows how tested methods perform across different dimensions and numbers of peaks. The results of ROOT-PV, ROOT-TFV, FTmPSO, RFTmPSO-s4, RAmQSO-s4 and RmNAFSA-s4 are summarized in Table VII.

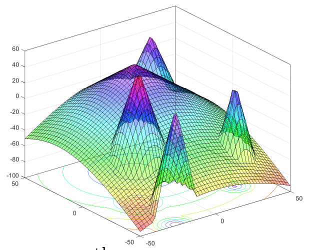
Not surprisingly, the average survival time of ROOT-TFV is better than that of ROOT-PV in all tests since ROOT-TFV eliminates predictor errors by assuming perfect knowledge of peak movements. Also, the autoregressive model, used by ROOT-PV as predictor [8], uses true values of solutions fitness values in previous environments for training. In a practical application where such information is not available, the performance of both algorithms will likely be worse.

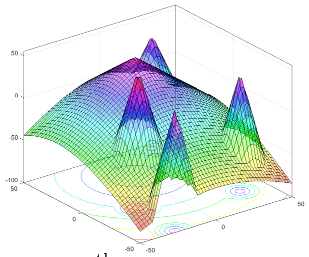
Figure 1 compares the true and predicted landscapes in D=2. Each environment is produced by 2500 points, and the parameter setting of mMPBR is based on default values in Table I with m=5. The first 15 environments are used to train the predictor [6], [8]. Figure 1 shows that the error of the predictor is noticeable even though the true fitness values in previous environments are used to train it.

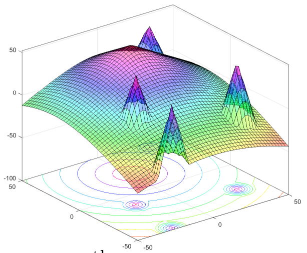
As can be seen in Table VII, ROOT-TFV has the highest average survival time in test instances with D=2 but loses its superiority in problems with D=5 and its performance, as well as that of ROOT-PV, experience a dramatic drop. To understand why these two methods struggle with even moderately dimensional problems, one has to note that they use (8) as fitness instead of the true fitness function (22). Figure 2 visualizes an example of the search space according to (8) in D=2. Figure 2(a) shows the true fitness landscape according to (22) and Fig. 2(b) shows the corresponding environment based on (8) with V=40 and its true five future environments. As can be seen, most of the problem landscapes defined by (8) are flat with a few narrow peaks. This is really challenging for the optimizer, especially in higher dimensions.

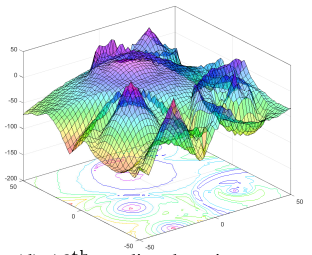
To investigate the performance of PSO in this type of environments, we use PSO for optimizing the mMPBR with 5 peaks and 100 environmental changes. This experiment is done 50 times and at the end of each environment, the $Gbest$ value of PSO based on the environment made by (8) is saved. The average $Gbest$ values are reported in Table VIII. Experiments for Table VIII are done on mMPBR in 2, 5 and 10 dimensions and with the number of evaluations per change f of 2500 and 10000 and $V = 40$.

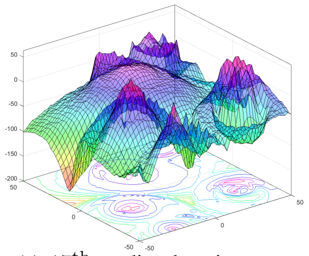
For f=2500 we used PSO with 50 particles and for f=10000 we increased the population size to 100. With D=2, although the second PSO benefits from a larger population size and more time to do exploration and exploitation in each environment, its performance is not so much better than the first PSO. With D=5, the performance of both PSOs decreases


(a) 16th true environment


(b) 17th true environment


(c) 18th true environment


(d) 16th predicted environment


(e) 17th predicted environment

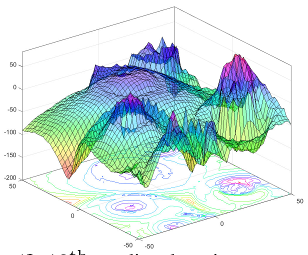
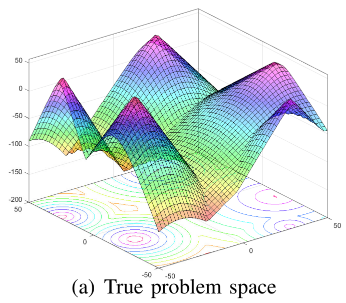

(f) 18th predicted environment

Fig. 1. An example of mMPBR in dimension D=2 to show the error of the predictor.


(a) True problem space

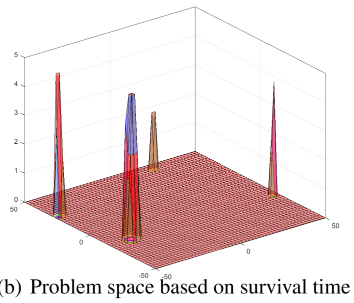

(b) Problem space based on survival time metric

Fig. 2. The search space made by (8) with a threshold V=40, dimension D=2 and peak number m=5 versus the true problem space.

TABLE VIII
AVERAGE GBEST VALUE (STANDARD ERROR IN PARENTHESIS) OF PSO IN SEARCH SPACE MADE BY (8) WITH DIFFERENT DIMENSION D.

Parameter settings	D=2	D=5	D=10
Population size=50, f=2500	5.02(0.17)	2.78(0.32)	0(0)
Population size=100, f=10000	5.31(0.16)	3.42(0.27)	0(0)

significantly and the results show that they are not able to find the best peak. Furthermore, the difference between the first and the second PSO increases relatively to their results in $D = 2$. This shows that the PSO needs more particles and time to deal with this type of environment. PSO fails to find peaks in $D = 10$.

Given the results in Table VIII and the fact that the search environment is shaped by the survival time metric (8) (an example is shown in Fig. 2(b)), we conclude that with increasing dimension, the search space becomes very challenging for optimizers using the survival time metric (8). This was confirmed in [11] where ROOT methods based on (8) have poor performance in higher dimensions. Our analysis provides an explanation for this behavior.

Table VII shows that the performance of FTmPSO, designed for TMO but used as ROOT algorithm, is better than ROOT-PV in most test instances and works surprisingly well at finding robust solutions. The only other paper that has investigated the performance of population-based algorithms designed for TMO in the context of ROOT is [6], and according to the reported results and analysis in this paper, some TMO algorithms also succeeded in finding robust solutions in ROOT. The reason behind the acceptable performance of some TMO based algorithms in ROOT is that in most research in the DOP domain, researchers have been working on DOPs with small changes, where the obtained knowledge from the current environment is useful for improving the optimization process in the next environment. In this type of environments which were also used in most ROOT papers, solutions around the peak centers can be robust solutions. Indeed, when comparing Fig. 2(a) and Fig. 2(b), it can be seen that robust solutions are around peak centers.

As can be seen in Table VII, the methods based on the proposed ROOT framework with strategy four can perform really well in maximizing the average survival time of robust solutions. All of RFTmPSO-s4, RAmQSO-s4 and RmNAFSA-s4 outperform ROOT-PV in all test instances in this section and only ROOT-TFV [8] (which, as mentioned before, is an unrealistic version of the ROOT algorithm due to its assumed knowledge of future environments) has better results in test problems with D=2. The average fitness value (11) of robust solutions obtained by the TMO algorithm is the best in all test instances because this algorithm chooses the best found solution in terms of fitness value.

For all three methods based on the proposed ROOT framework, the average fitness value of robust solutions in all test instances are better than that of ROOT-PV and ROOT-TFV because the proposed methods search the problem space with actual fitness values and choose one of the peaks as robust solution. On the other hand, ROOT-PV and ROOT-TFV use the survival time metric and thus can get stuck in flat areas (Fig. 2(b)). For the same reason, their average fitness value can be very poor in problems with higher D and V (e.g. these two algorithms achieve negative average fitness values in D=5, m=5).

In this section, we embedded three different multi-swarm methods in the proposed ROOT framework. The reported results in Table VII show that the proposed algorithms are able to perform better than previous state-of-the-art survival time metric (8) based methods especially on the environments with higher number of dimensions.

Comparing results of RFTmPSO-s4, RAmQSO-s4 and RmNAFSA-s4, we realize that the quality of swarms in finding and tracking peaks can improve the proposed frameworks performance noticeably. Specifically, better peak finding and tracking performance corresponds to more accurate information (gathered by (15), (16) and (17)) leading to more reliable decision making by (18), (19), (20) and (21).

VI. Conclusion

A new framework for robust optimization over time (ROOT) was proposed. In the proposed framework, a multi-swarm/multi-population method is responsible for finding, tracking and monitoring peaks. Each sub-swarm gathers information about its covered peak. This information is used to predict the future behavior of the peak and pick the next robust solution in case the current solution becomes unacceptable. We used three types of information based on shift severity, height variance and fitness variance of peaks and designed four different solution selection strategies based on this information. The experimental results show that the fourth strategy that uses the information about shift severity as well as height variance of peaks had the best performance overall and can be used for other problem instances.

We used a wide range of problem settings to investigate the performance of the proposed framework based algorithms versus the existing state-of-the-art framework based on a survival time metric. We showed that the performance of previous methods that use the survival time metric is substantially worse in problems with higher dimensions. All previous state-of-the-art methods attempt to predict future fitness values of solutions based on previous fitness values of solutions. However, this is a difficult task and can become almost impossible for problems with higher dimensions, larger search space and higher change frequencies. In our experiments, we thus investigate the considerable effect of predictor errors and approximation errors on the performance of previous methods. On the other hand, our proposed framework does not have to deal with the challenges of predicting future fitness values. The experimental results show that the performance of the proposed framework is significantly better than that of state-of-the-art methods especially in problems with higher dimensions.

We tested the proposed framework based algorithms on problem instances with different combinations of parameter settings of mMPBR and provided performance analysis based on them. The results showed that the problem becomes more challenging when shift severities of peaks, dimension of problem space, and change frequency are higher. However, the reported results showed that the proposed methods were able to perform very well even in more challenging problems.

Future work will include a study of other peaks behavioral information and design of new strategies for choosing robust solutions. Additionally, other objectives of robust optimization over time such as minimizing solution change cost will be investigated. Another interesting area is ROOT for DOPs with undetectable changes [36]. Finally, we will investigate the performance of the proposed framework on other types of problems including real-world applications.

Acknowledgements

The authors would like to thank Dr. Mohammad Nabi Omidvar for the valuable discussions and his constructive feedback.

References

[1] 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, 2012.

[2] D. E.Wilkins, S. F.Smith, L. A.Kramer, T. J.Lee, and T. W.Rauenbusch, "Airlift mission monitoring and dynamic rescheduling," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 2, pp. 141–155, 2008.

[3] J. A. D. Atkin, E. K. Burke, J. S. Greenwood, and D. Reeson, "On-line decision support for take-off runway scheduling with uncertain taxi times at london heathrow airport," *Journal of Scheduling*, vol. 11, no. 5, pp. 323–346, 2008.

[4] X. Yu, Y. Jin, K. Tang, and X. Yao, "Robust optimization over time - a new perspective on dynamic optimization problems," in *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2010, pp. 1–6.

[5] H. Fu, B. Sendhoff, K. Tang, and X. Yao, "Characterizing environmental changes in robust optimization over time," in *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2012, pp. 1–8.

[6] Y. Jin, K. Tang, X. Yu, B. Sendhoff, and X. Yao, "A framework for finding robust optimal solutions over time," *Memetic Computing*, vol. 5, no. 01, pp. 3–18, 2013.

[7] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *International Conference on Neural Networks*, vol. 04. IEEE, 1995, pp. 1942–1948.

[8] H. Fu, B. Sendhoff, K. Tang, and X. Yao, "Finding robust solutions to dynamic optimization problems," in *Applications of Evolutionary Computation*, vol. 7835. Lecture Notes in Computer Science, 2013, pp. 616–625.

[9] H. Fu, B. Sendhoff, K. Tang, and X. Yao, "Robust optimization over time: problem difficulties and benchmark problems," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 5, pp. 731–745, 2015.

[10] Y. nan Guo, M. Chen, H. Fu, and Y. Liu, "Find robust solutions over time by two-layer multi-objective optimization method," in *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2014, pp. 1528–1535.

[11] Y. Huang, Y. Ding, K. Hao, and Y. Jin, "A multi-objective approach to robust optimization over time considering switching cost," *Information Sciences*, vol. 394-395, pp. 183–197, 2017.

[12] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. Wiley, 2015.

[13] M. Mavrovouniotis, C. Li, and S. Yang, “A survey of swarm intelligence for dynamic optimization: Algorithms and applications,” *Swarm and Evolutionary Computation*, vol. 33, pp. 1–17, 2017.
[14] J. Branke, T. Kaussler, C. Smidt, and H. Schmeck, “A multipopulation approach to dynamic optimization problems,” in *Evolutionary Design and Manufacture*, 2000, pp. 299–307.
[15] D. Yazdani, B. Nasiri, R. Azizi, A. Sepas-Moghaddam, and M. R. Meybodi, “Optimization in dynamic environments utilizing a novel method based on particle swarm optimization,” *International Journal of Artificial Intelligence*, vol. 11, pp. 170–192, 2013.
[16] C. Li and S. Yang, “Optimization in dynamic environments utilizing a novel method based on particle swarm optimization,” in *4th International Conference on Natural Computation*. IEEE, 2008, pp. 624–628.
[17] A. Sepas-Moghaddam, A. Arabshahi, D. Yazdani, and M. M. Dehshibi, “A novel hybrid algorithm for optimization in multimodal dynamic environments,” in *International Conference on Hybrid Intelligent Systems (HIS)*. IEEE, 2012, pp. 143–148.
[18] B. Nasiri, M. R. Meybodi, and M. M. Ebadzadeh, “History-driven particle swarm optimization in dynamic and uncertain environments,” *Neurocomputing*, vol. 172, pp. 356–370, 2016.
[19] D. Yazdani, A. Sepas-Moghaddam, A. Dehban, and N. Horta, “A novel approach for optimization in dynamic environments based on modified artificial fish swarm algorithm,” *International Journal of Computational Intelligence and Applications*, vol. 15, no. 02, pp. 1 650 010–1 650 034, 2016.
[20] D. Yazdani, B. Nasiri, A. Sepas-Moghaddam, M. R. Meybodi, and M. Akbarzadeh-Totonchi, “mNAFSA: A novel approach for optimization in dynamic environments with global changes,” *Swarm and Evolutionary Computation*, vol. 18, pp. 38–53, 2014.
[21] D. Yazdani, M. R. Akbarzadeh-Totonchi, B. Nasiri, and M. R. Meybodi, “A new artificial fish swarm algorithm for dynamic optimization problems,” in *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2012, pp. 1–8.
[22] 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. 04, pp. 440–458, 2006.
[23] S. Bird and X. Li, “Using regression to improve local convergence,” in *IEEE Congress on Evolutionary Computation*. IEEE, 2007, pp. 592–599.
[24] C. Li and S. Yang, “A clustering particle swarm optimizer for dynamic optimization,” in *IEEE Congress on Evolutionary Computation*. IEEE, 2009, pp. 439–446.
[25] S. Yang and C. Li, “A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments,” *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 06, pp. 959–974, 2010.
[26] W. Du and B. Li, “Multi-strategy ensemble particle swarm optimization for dynamic optimization,” *Information Sciences*, vol. 178, no. 15, pp. 3096–3109, 2008.
[27] 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, 2006.
[28] T. Blackwell, J. Branke, and X. Li, “Particle swarms for dynamic optimization problems,” in *Swarm Intelligence: Introduction and Applications*, C. Blum and D. Merkle, Eds. Springer, 2008, pp. 193–217.
[29] C. Li, T. T. Nguyen, M. Yang, M. Mavrovouniotis, and S. Yang, “An adaptive multi-population framework for locating and tracking multiple optima,” *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 05, pp. 590–605, 2016.
[30] D. Yazdani, B. Nasiri, A. Sepas-Moghaddam, and M. R. Meybodi, “A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization,” *Applied Soft Computing*, vol. 13, no. 04, pp. 2144–2158, 2013.
[31] D. Yazdani, T. T. Nguyen, J. Branke, and J. Wang, “A multi-objective time-linkage approach for dynamic optimization problems with previous-solution displacement restriction,” in *European Conference on the Applications of Evolutionary Computation*, K. Sim and P. Kaufmann, Eds. Lecture Notes in Computer Science, 2018, vol. 10784, pp. 864–878.
[32] T. T. Nguyen, “Continuous dynamic optimisation using evolutionary algorithms,” Ph.D. dissertation, University of Birmingham, Birmingham, UK, 2011.
[33] J. Branke, “Memory enhanced evolutionary algorithms for changing optimization problems,” in *IEEE Congress on Evolutionary Computation*. IEEE, 1999, pp. 1875–1882.
[34] D. Yazdani, T. T. Nguyen, J. Branke, and J. Wang, “A new multi-swarm particle swarm optimization for robust optimization over time,” in *Applications of Evolutionary Computation*, G. Squillero and K. Sim, Eds. Springer Lecture Notes in Computer Science, 2017, vol. 10200, pp. 99–109.
[35] C. Li, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, and H. G. Beyer, “Benchmark generator for cec 2009 competition on dynamic optimization,” Department of Computer Science, University of Leicester, UK, Tech. Rep., 2009.
[36] C. Li and S. Yang, “A general framework of multipopulation methods with clustering in undetectable dynamic environments,” *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 4, pp. 556–577, 2012.

Danial Yazdani received his BSc in 2007 from Shirvan Azad University, and his MSc in 2011 from Qazvin Azad University in computer science. Currently, he is a PhD student at Liverpool John Moores University. His main research interests include different types of dynamic optimization problems such as constrained, time-linkage, multi-objective, large-scale, and combinatorial. He has published more than 20 papers in peer-reviewed journals and conferences.

Trung Thanh Nguyen received his BSc in 2000 from Vietnam National University, and his MPhil and PhD in Computing Science from University of Birmingham in 2007 and 2011, respectively. He has been a Reader in Operational Research at Liverpool John Moores University (LJMU) since 2015. Prior to that, he was a Senior Lecturer in Optimisation and Simulation Modelling at LJMU since 2013, and a Research Fellow at LJMU and University of Birmingham in 2011. His current research interests include operational research/dynamic optimisation with a particular application to logistics/transport problems.

He is currently the principal investigator of four research grants in transport and logistics. He has published more than 40 peer-reviewed papers. He is/was the chair of three leading conference tracks, member of TPCs of over 30 leading conferences, editor of five books, two journals and invited speaker of various conferences and events.

Jürgen Branke (M’02) received the Ph.D. degree from University of Karlsruhe, Karlsruhe, Germany, in 2000. He is a Professor of Operational Research and Systems with the Warwick Business School, University of Warwick, Coventry, U.K. He has been an active researcher in the area of evolutionary optimization since 1994 and has published more than 170 papers in international peer-reviewed journals and conferences. His research interests include multiobjective optimization, handling of uncertainty in optimization, dynamically changing optimization problems, simulation-based optimisation and the design of complex systems. He is area editor of the Journal of Heuristics and the Journal on Multi-Criteria Decision Analysis, as well as associate editor of IEEE Transactions in Evolutionary Computation and the Evolutionary Computation Journal.

Supplementary Document of 'Robust optimization over time by learning problem space characteristics'

Danial Yazdani, Trung Thanh Nguyen, and Jürgen Branke

CONTENTS

S-I. SENSITIVITY ANALYSIS

In this document, we investigate the effect of different parameter settings of FTmPSO [1] as a multi-swarm method embedded in the proposed ROOT framework on the average survival time. To test the sensitivity to a particular parameter, we change this parameter while keeping all other parameters as specified in Table II. Moreover, we investigate the effect of different population sizes on the performance of the ROOT-PV method. Experiments are done on mMPBR with its default parameter setting reported in Table I. All experimental results are obtained by performing 30 independent runs. Best results based on Wilcoxon signed-rank test with Holm-Bonferroni p-value correction, $\alpha = 0.05$ are highlighted in each table.

A. Effect of FTmPSO parameter settings on RFTmPSO performance

The first set of experiments examines the effect of the tracker-swarms population size on the obtained survival time by all four versions of RFTmPSO which is reported in Table S-I. Overall, results demonstrate that five-particle tracker-swarms are best. According to Section III-A, the multi-swarm algorithm in the proposed ROOT framework needs to track multiple optima (TMO), which is similar to its original purpose. If the multi-swarm method tracks peaks properly, more accurate information can be provided for the phase of selecting more robust solutions.

Table S-II illustrates the obtained results from algorithms with different number of particles in the finder-swarm (a sub-swarm in FTmPSO that is responsible for finding uncovered peaks). As it can be observed, the best performance overall is obtained when the finder-swarms population size is 10. Lower values result in decreasing ability of this sub-swarm to find uncovered peaks. On the other hand, a higher population size of the finder-swarm results in a waste of computational resources (fitness evaluations). Our deeper analysis using visual plots indicated that a larger finder-swarm leads to a convergence to better peaks, due to an increase in exploration ability. Consequently, smaller peaks may not be detected until they become larger which leads to a decrease in the accuracy of the gathered information by FTmPSO.

TABLE S-I
THE OBTAINED AVERAGE SURVIVAL TIME (AND STANDARD ERROR) FROM THE RFTMPSO ALGORITHMS WITH DIFFERENT SUB-SWARM'S POPULATION SIZE (TPS) ON THE DEFAULT SCENARIO OF MMPBR.

V	TPS	RFTmPSO-s1	RFTmPSO-s2	RFTmPSO-s3	RFTmPSO-s4
40	3	5.12(0.49)	5.25(0.61)	4.54(0.42)	5.82(0.47)
	4	5.53(0.35)	5.56(0.62)	4.53(0.36)	6.16(0.68)
	5	5.48(0.60)	5.60(0.82)	4.89(0.81)	6.14(0.85)
	7	5.17(0.73)	5.54(0.86)	4.76(0.76)	6.19(0.88)
	10	4.95(0.37)	5.02(0.67)	4.52(0.62)	5.16(1.28)
50	3	3.61(0.43)	3.65(0.46)	3.33(0.39)	3.84(0.33)
	4	3.90(0.32)	3.59(0.31)	3.52(0.31)	4.05(0.39)
	5	3.90(0.34)	3.63(0.33)	3.44(0.34)	4.22(0.41)
	7	3.81(0.40)	3.66(0.50)	3.36(0.60)	4.23(0.49)
	10	3.34(0.32)	3.41(0.40)	3.26(0.42)	4.17(0.78)
50	3	2.33(0.24)	2.26(0.24)	2.29(0.20)	2.71(0.20)
	4	2.36(0.20)	2.12(0.15)	2.36(0.23)	2.77(0.26)
	5	2.55(0.21)	2.43(0.26)	2.51(0.28)	2.77(0.31)
	7	2.62(0.40)	2.21(0.41)	2.50(0.44)	2.83(0.24)
	10	2.19(0.27)	2.42(0.26)	2.14(0.28)	2.66(0.30)

TABLE S-II
THE OBTAINED AVERAGE SURVIVAL TIME (AND STANDARD ERROR) FROM THE RFTMPSO ALGORITHMS WITH DIFFERENT FINDER-SWARMS POPULATION SIZE (FPS) ON THE DEFAULT SCENARIO OF MMPBR.

V	FPS	RFTmPSO-s1	RFTmPSO-s2	RFTmPSO-s3	RFTmPSO-s4
40	5	4.83(0.44)	4.95(0.36)	4.21(0.33)	5.28(0.36)
	7	5.01(0.50)	5.35(0.55)	4.71(0.49)	5.90(0.48)
	10	5.48(0.60)	5.60(0.82)	4.89(0.81)	6.14(0.85)
	12	5.39(0.44)	5.30(0.39)	4.59(0.35)	5.82(0.55)
	15	5.26(0.60)	5.18(0.59)	4.47(0.42)	5.45(0.47)
45	5	3.08(0.28)	3.16(0.24)	3.13(0.31)	3.64(0.85)
	7	3.52(0.40)	3.49(0.40)	3.40(0.35)	4.29(0.37)
	10	3.90(0.34)	3.63(0.33)	3.44(0.34)	4.22(0.41)
	12	3.89(0.30)	3.62(0.27)	3.34(0.31)	4.04(0.30)
	15	3.34(0.56)	3.40(0.35)	3.29(0.31)	3.90(0.37)
50	5	2.01(0.19)	1.97(0.18)	2.08(0.19)	2.16(0.41)
	7	2.63(0.34)	2.33(0.32)	2.65(0.30)	2.56(0.28)
	10	2.55(0.21)	2.43(0.26)	2.51(0.28)	2.77(0.31)
	12	2.45(0.15)	2.47(0.20)	2.46(0.18)	2.59(0.24)
	15	2.25(0.47)	3.10(0.47)	2.58(0.32)	2.57(0.28)

Table S-III shows the effect of different values of the *Conv-limit*, parameter which is used for determining finder-swarms convergence. According to the results presented in Table S-III, decreasing the values of *Conv-limit* leads to decreasing the performance of algorithms because the finder-swarms convergence condition will not be met in an appropriate time. Therefore, it will take more time for the finder-swarm to create a tracker-swarm on the peak and continue its search for finding other possible uncovered peaks. By increasing the value of *Conv-limit* up to 1, the algorithms' efficiency increases. However, increasing its value beyond 1 will deteriorate performance, probably because a high value of *Conv-limit* results the finder-swarm being considered converged very early, which leads to the creation of unnecessary tracker-swarms, wasting computational resources.

TABLE S-III
THE OBTAINED AVERAGE SURVIVAL TIME (AND STANDARD ERROR) FROM THE RFTMPSO ALGORITHMS WITH DIFFERENT *Conv-limit* (CL) ON THE DEFAULT SCENARIO OF MMPBR.

V	CL	RFTmPSO-s1	RFTmPSO-s2	RFTmPSO-s3	RFTmPSO-s4
	0.1	4.28(0.51)	4.84(0.63)	4.47(0.72)	4.88(0.71)
	0.5	4.53(0.45)	5.19(0.47)	4.90(0.44)	5.38(0.47)
40	1	4.89(0.81)	5.60(0.82)	5.48(0.60)	6.14(0.85)
	2	4.85(0.52)	5.63(0.71)	5.42(0.53)	5.80(0.69)
	5	4.51(0.54)	5.23(0.50)	5.06(0.77)	5.27(0.57)
	0.1	3.18(0.29)	3.13(0.37)	3.25(0.42)	3.80(0.43)
	0.5	3.23(0.32)	3.39(0.26)	3.51(0.31)	4.04(0.30)
45	1	3.44(0.34)	3.63(0.33)	3.90(0.34)	4.22(0.41)
	2	3.24(0.41)	3.66(0.46)	3.55(0.38)	3.98(0.53)
	5	3.01(0.46)	3.06 (0.48)	3.23(0.61)	3.78(0.52)
	0.1	2.24(0.24)	2.21(0.26)	2.09(0.27)	2.32(0.29)
	0.5	2.38(0.21)	2.43(0.20)	2.40(0.27)	2.81(0.24)
50	1	2.51(0.28)	2.43(0.26)	2.55(0.21)	2.77(0.31)
	2	2.32(0.26)	2.40(0.30)	2.51(0.30)	2.75(0.30)
	5	2.26(0.33)	2.15(0.32)	1.98(0.40)	2.21(0.43)

Another parameter involved in finder-swarm convergence determination is K. The effect of its different values on the algorithms' performance is shown in Table S-IV. Similar to *Conv-limit*, lower values of K result in creating more unnecessary tracker-swarms whereas higher values delay the finder-swarm convergence determination. According to Table S-IV, best performance is generally obtained with $K = 10$.

TABLE S-IV
THE OBTAINED AVERAGE SURVIVAL TIME (AND STANDARD ERROR) FROM THE RFTMPSO ALGORITHMS WITH DIFFERENT K ON THE DEFAULT SCENARIO OF MMPBR.

V	K	RFTmPSO-s1	RFTmPSO-s2	RFTmPSO-s3	RFTmPSO-s4
	5	4.71(4.06)	5.24(0.68)	5.29(0.84)	6.00(0.69)
	7	4.83(0.37)	5.40(0.45)	5.32(0.41)	6.07(0.50)
40	10	4.89(0.81)	5.60(0.82)	5.48(0.60)	6.14(0.85)
	12	4.90(0.47)	5.31(0.47)	5.53(0.54)	5.90(0.54)
	15	4.71(0.45)	5.28(0.52)	5.04(0.40)	5.58(0.66)
	5	3.06(0.50)	3.35(0.55)	3.31(0.34)	3.95(0.60)
	7	3.16(0.27)	3.65(0.29)	3.52(0.27)	3.90(0.41)
45	10	3.44(0.34)	3.63(0.33)	3.90(0.34)	4.22(0.41)
	12	3.26(0.44)	3.45(0.43)	4.05(0.48)	4.03(0.40)
	15	3.40(0.32)	3.34(0.27)	3.62(0.23)	3.84(0.51)
	5	2.32(0.36)	2.33(0.36)	2.37(0.34)	2.75(0.32)
	7	2.20(0.20)	2.54(0.21)	2.42(0.21)	2.81(0.20)
50	10	2.51(0.28)	2.43(0.26)	2.55(0.21)	2.77(0.31)
	12	2.29(0.20)	2.59(0.22)	2.51(0.22)	2.75(0.27)
	15	1.95(0.14)	2.12(0.16)	1.99(0.17)	2.66(0.30)

P and Q control the diversity introducing of tracker-swarms after environmental changes. On the one hand, lower values of P and Q result in a lower initial diversity of tracker-swarms at the beginning of each environment which leads to a decrease in their tracking ability. On the other hand, higher values of these parameters cause over-diversification of tracker-swarms which leads to increasing the possibility of migrating tracker-swarms to other peaks when peaks are very close to each other. Additionally, over-diversification decreases the exploitation ability. The results of using different values of P and Q are reported in Tables S-V and S-VI. Note that, although in [1] P and Q work based on the shift severity of peaks that was available for FTmPSO as an initial knowledge, here algorithms have to learn about peaks shift severities by themselves using (15).

TABLE S-V
THE OBTAINED AVERAGE SURVIVAL TIME (AND STANDARD ERROR) FROM THE RFTMPSO ALGORITHMS WITH DIFFERENT Q ON THE DEFAULT SCENARIO OF MMPBR.

V	Q	RFTmPSO-s1	RFTmPSO-s2	RFTmPSO-s3	RFTmPSO-s4
	0.5	5.20(0.37)	5.55(0.46)	5.15(0.56)	6.00(0.69)
	0.75	5.39(0.51)	5.30(0.60)	5.10(0.74)	6.17(0.68)
40	1	5.48(0.60)	5.60(0.82)	4.89(0.81)	6.14(0.85)
	1.5	5.15(0.40)	5.22(0.52)	4.93(0.70)	5.67(0.75)
	2	5.02(0.53)	4.97(0.83)	4.59(0.73)	5.14(0.81)
	0.5	3.65(0.32)	3.48(0.34)	3.21(0.33)	4.14(0.34)
	0.75	3.71(0.40)	3.60(0.39)	3.36(0.41)	4.27(0.40)
45	1	3.90(0.34)	3.63(0.33)	3.44(0.34)	4.22(0.41)
	1.5	3.91(0.25)	3.68(0.45)	3.18(0.39)	4.31(0.46)
	2	3.45(0.44)	3.25(0.44)	3.19(0.53)	4.12(0.48)
	0.5	2.36(0.26)	2.29(0.24)	2.25(0.27)	2.38(0.25)
	0.75	2.56(0.21)	2.35(0.22)	2.56(0.22)	2.72(0.23)
50	1	2.55(0.21)	2.43(0.26)	2.51(0.28)	2.77(0.31)
	1.5	2.58(0.26)	2.39(0.20)	2.49(0.26)	2.63(0.26)
	2	2.28(0.26)	2.23(0.27)	2.20(0.28)	2.26(0.26)

TABLE S-VI
THE OBTAINED AVERAGE SURVIVAL TIME (AND STANDARD ERROR) FROM THE RFTMPSO ALGORITHMS WITH DIFFERENT P ON THE DEFAULT SCENARIO OF MMPBR.

V	P	RFTmPSO-s1	RFTmPSO-s2	RFTmPSO-s3	RFTmPSO-s4
	0.5	5.24(0.35)	5.62(0.51)	4.58(0.56)	5.91(0.48)
	0.75	5.40(0.55)	5.46(0.56)	4.70(0.60)	5.61(0.57)
40	1	5.48(0.60)	5.60(0.82)	4.89(0.81)	6.14(0.85)
	2	5.49(0.46)	5.42(0.60)	5.14(0.66)	6.08(0.92)
	5	5.28(0.70)	5.18(0.86)	4.64(0.78)	5.87(0.54)
	0.5	3.68(0.30)	3.68(0.37)	3.22(0.35)	3.85(0.39)
	0.75	3.71(0.36)	3.53(0.37)	3.29(0.28)	4.15(0.34)
45	1	3.90(0.34)	3.63(0.33)	3.44(0.34)	4.22(0.41)
	2	3.88(0.38)	3.69(0.38)	3.31(0.55)	4.14(0.34)
	5	3.71(0.41)	3.61(0.45)	3.20(0.67)	3.79(0.54)
	0.5	2.33(0.29)	2.34(0.31)	2.56(0.26)	2.78(0.24)
	0.75	2.36(0.22)	2.49(0.27)	2.19(0.22)	2.39(0.26)
50	1	2.55(0.21)	2.43(0.26)	2.51(0.28)	2.77(0.31)
	2	2.38(0.19)	2.44(0.31)	2.50(0.35)	2.82(0.26)
	5	2.25(0.27)	2.32(0.33)	2.40(0.33)	2.75(0.28)

The last investigated parameter is $excl_{\text{factor}}$ which is used in (14). According to Table S-VII, for embedded multi-swarm methods in the proposed framework, this threshold needs to be lower (i.e. 0.1) in comparison with its value in the original references [2], [3] (i.e. 0.5), because we want to avoid losing information about a peak only because it moves close to another peak. Therefore, sub-swarms need to be closer to be involved in exclusion. In fact, higher values of $excl_{\text{factor}}$ increase the possibility of involving sub-swarms whose under covered peaks are close to each other in exclusion condition. Consequently, the algorithm may lose valuable information about a peak by removing a sub-swarm by exclusion mechanism.

TABLE S-VII
THE OBTAINED AVERAGE SURVIVAL TIME (AND STANDARD ERROR) FROM THE RFTMPSO ALGORITHMS WITH DIFFERENT $excl_{\text{factor}}$ (EF) ON THE DEFAULT SCENARIO OF MMPBR.

V	EF	RFTmPSO-s1	RFTmPSO-s2	RFTmPSO-s3	RFTmPSO-s4
40	0.05	5.13(0.55)	5.21(0.68)	4.32(0.71)	5.52(0.72)
	0.1	5.48(0.60)	5.60(0.82)	4.89(0.81)	6.14(0.85)
	0.25	5.23(0.48)	5.26(0.65)	4.58(0.60)	5.70(0.54)
	0.5	5.47(0.54)	5.35(0.79)	4.83(0.68)	5.86(0.79)
	1	4.86(1.11)	5.08(0.71)	4.39(1.07)	4.47(1.07)
45	0.05	3.63(0.40)	3.67(0.51)	3.31(0.57)	4.27(0.51)
	0.1	3.90(0.34)	3.63(0.33)	3.44(0.34)	4.22(0.41)
	0.25	3.52(0.31)	3.67(0.36)	3.28(0.38)	3.91(0.39)
	0.5	3.55(0.42)	3.70(0.51)	3.62(0.46)	4.07(0.47)
	1	3.31(0.69)	3.35(0.71)	3.13(0.60)	3.05(0.60)
50	0.05	2.34(0.28)	2.47(0.29)	2.24(0.37)	2.20(0.31)
	0.1	2.55(0.21)	2.43(0.26)	2.51(0.28)	2.77(0.31)
	0.25	2.23(0.29)	2.31(0.30)	2.54(0.23)	2.32(0.27)
	0.5	2.55(0.24)	2.43(0.32)	2.37(0.20)	2.00(0.32)
	1	2.02(0.51)	2.04(0.55)	1.92(0.54)	1.93(0.54)

B. *Effect of different parameter settings of ROOT-PV on its performance*

In this part, we investigate the effect of the population size on the performance of ROOT-PV, which is the more realistic version of Fu's method [4]. Since Fu's method [4] is a single-swarm method based on the survival time metric in (8), population size of the PSO is the most important parameter. Table S-VIII shows the effect of different population sizes (PS) on the obtained average survival time by ROOT-PV. As suggested in [4], best results are obtained when population size is 50.

TABLE S-VIII
OBTAINED AVERAGE SURVIVAL TIME (AND STANDARD ERROR) BY ROOT-PV WITH DIFFERENT POPULATION SIZE ON MMPBR WITH ITS DEFAULT PARAMETER SETTINGS REPORTED IN TABLE I.

PS	$V = 40$	$V = 45$	$V = 50$
10	1.47(0.28)	0.80(0.18)	0.24(0.10)
20	1.10(0.17)	1.00(0.24)	0.27(0.07)
30	1.85(0.22)	0.62(0.10)	0.32(0.08)
40	2.11(0.37)	0.80(0.16)	0.30(0.07)
50	2.26(0.61)	1.11(0.23)	0.37(0.23)
60	1.84(0.20)	1.10(0.34)	0.31(0.19)
75	1.83(0.47)	0.94(0.19)	0.29(0.10)
100	1.67(0.24)	0.69(0.15)	0.22(0.07)

REFERENCES

[1] D. Yazdani, B. Nasiri, A. Sepas-Moghaddam, and M. R. Meybodi, "A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization," *Applied Soft Computing*, vol. 13, no. 04, pp. 2144–2158, 2013.

[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, 2006.

[3] T. Blackwell, J. Branke, and X. Li, "Particle swarms for dynamic optimization problems," in *Swarm Intelligence: Introduction and Applications*, C. Blum and D. Merkle, Eds. Springer, 2008, pp. 193–217.

[4] H. Fu, B. Sendhoff, K. Tang, and X. Yao, "Finding robust solutions to dynamic optimization problems," in *Applications of Evolutionary Computation*, vol. 7835. Lecture Notes in Computer Science, 2013, pp. 616–625.