Honda Research Institute EU
Honda Research Institute Europe GmbH
https://www.honda-ri.de/

Robust Optimization Over Time: Problem Difficulties and Benchmark Problems

Haobo Fu, Bernhard Sendhoff, Ke Tang, Xin Yao

2015

Preprint:

This is an accepted article published in IEEE Transactions on Evolutionary Computation. The final authenticated version is available online at: https://doi.org/[DOI not available]

Robust Optimization Over Time: Problem Difficulties and Benchmark Problems

Haobo Fu, Bernhard Sendhoff, *Senior Member, IEEE*, Ke Tang, *Senior Member, IEEE*, and Xin Yao *Fellow, IEEE*

***Abstract*—The focus of most research in evolutionary dynamic optimization has been tracking moving optimum (TMO). Yet, TMO does not capture all the characteristics of real-world dynamic optimization problems (DOPs), especially in situations where a solution's future fitness has to be considered. To account for a solution's future fitness explicitly, we propose to find robust solutions to DOPs, which are formulated as the robust optimization over time (ROOT) problem. In this paper we analyze two robustness definitions in ROOT and then develop two types of benchmark problems for the two robustness definitions in ROOT, respectively. The two types of benchmark problems are motivated by the inappropriateness of existing DOP benchmarks for the study of ROOT. Additionally, we evaluate four representative methods from the literature on our proposed ROOT benchmarks, in order to gain a better understanding of ROOT problems and their relationship to more popular TMO problems. The experimental results are analyzed, which show the strengths and weaknesses of different methods in solving ROOT problems with different dynamics. In particular, the real challenges of ROOT problems have been revealed for the first time by the experimental results on our proposed ROOT benchmarks.**

***Index Terms*—Benchmarking, dynamic optimization problems (DOPs), evolutionary algorithms (EAs), robust optimization over time (ROOT).**

Manuscript received January 26, 2013; revised October 08, 2013, February 02, 2014, and June 19, 2014; accepted November 10, 2014. Date of publication December 4, 2014; date of current version September 29, 2015. This work was supported in part by Honda Research Institute Europe, in part by the Engineering and Physical Sciences Research Council under Grant EP/K001523/1, in part by the EU FP7 International Research Staff Exchange Scheme under Grant 247619, in part by the National Natural Science Foundation of China under Grants 61329302 and 61175065, in part by the Program for New Century Excellent Talents in University under Grant NCET-12-0512, and in part by the Science and Technological Fund of Anhui Province for Outstanding Youth under Grant 1108085J16. The work of X. Yao was supported by the Royal Society Wolfson Research Merit Award.

H. Fu is with the Centre of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K. (e-mail: hxf990@cs.bham.ac.uk).

B. Sendhoff is with Honda Research Institute Europe, Offenbach 63073, Germany.

K. Tang is with University of Science and Technology of China (USTC)–Birmingham Joint Research Institute in Intelligent Computation and its Applications, School of Computer Science and Technology, USTC, Hefei 230027, China.

X. Yao is with the Centre of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K., and also with the University of Science and Technology of China (USTC)–Birmingham Joint Research Institute in Intelligent Computation and its Applications, School of Computer Science and Technology, USTC, Hefei 230027, China.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2014.2377125

I. Introduction

DYNAMIC optimization problems (DOPs) are optimization problems that are changing over time. The change can happen in the objective fitness function and the number of design variables, as well as the constraints. The research on applying evolutionary algorithms (EAs) to DOPs has become very active [1]–[3]. On the one hand, this is due to the fact that most real-world optimization problems are inherently dynamic. On the other hand, EAs are believed to be suitable methods [1] for solving several DOPs as EAs maintain a population that may potentially be good at adapting to dynamic environments.

For static optimization problems, usually only one solution, which is found to be best in terms of fitness for the corresponding objective fitness function among all evaluated solutions, is determined and implemented[1] in practice. However, for DOPs, the decision maker has to repeatedly implement solutions over time in a changing environment. Accordingly, DOPs can be classified following the way the decision maker implements solutions in a changing environment. So far, the majority of research on applying EAs to DOPs has been focusing on one type of DOPs, i.e., tracking moving optimum (TMO) problems [4]–[8]. For TMO problems, the objective for the current time is to find an optimum in terms of fitness for the current environment and then relocate a new optimum in terms of fitness for the new environment once the environment changes. Therefore, it is implicitly assumed in TMO that the decision maker has to determine and implement a new solution every time the environment changes. The formalization of TMO problems is justifiable in situations where implementing a solution, e.g., updating some control variables, can be finished instantaneously and inexpensively, such as in [9]–[11]. However, in situations where the implementation of a solution involves human operation [12], resource transportation [13], etc., which all incur a huge cost, it might be impossible to implement a new solution every time the environment changes. In other words, TMO may be an inappropriate formalization of the corresponding DOP. This is further explained in the following.

1) A lot of real-world DOPs involve human operation. Taking practical dynamic vehicle routing problems for example [12], [14], [15], the environmental states,

[1]We would like to emphasize the difference between determination of a solution and implementation of a solution. Determination of a solution is completed once the algorithm finishes its computation, while implementation of a solution involves the practical operation of the determined solution.

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/

e.g., demands from customers, conditions of road, etc., vary from day to day. For this kind of DOPs, if we implement a different solution every day, it will cause disruptions to the working timetable of staff and may also confuse the operators as the operators would daily change their implementations [15]. Therefore, in such circumstances, it would be more beneficial having a fixed and good solution implemented and used for a long time period than implementing a new solution every time the environment changes, which is done in TMO.

2) In some circumstances, it is desirable to have an already implemented solution in use as much time as possible, during which environmental changes may happen, as long as the solution maintains its feasibility and its fitness above a certain level. Such circumstances can be found in aircraft taking-off/landing scheduling problems [16], [17]. As long as the system performance is maintained above a certain level, it is preferred to stick to an already implemented solution/schedule after an environmental change. The reason is that any modification of an already implemented solution will cause unfavorable disruptions to the operations in the airport, e.g., rescheduling some other aeroplanes. Therefore, in such circumstances, it is more practical to use a solution that can maintain its feasibility and its fitness above a threshold as long as possible, starting from the time when it is first implemented, than to use a solution that is determined by TMO, i.e., maximizing the fitness only for the current environment.

In order to account for the two aforementioned situations where TMO is not a proper formalization of DOPs, we proposed the idea of finding robust solutions to DOPs, and the corresponding formalization is referred to as robust optimization over time (ROOT) [18], [19].

To account for the first aforementioned situation where it is desirable to use a solution for a long time period, during which environmental changes occur, we proposed the following robustness definition to quantify the performance of a solution during a time period [20]. Assuming a solution $\mathbf{x}$ is determined at time t, the corresponding robustness is defined as

$$F^a(\mathbf{x}, t, T) = \frac{1}{T}\int_{t}^{t+T} f(\mathbf{x}, \boldsymbol{\alpha}(i))di \tag{1}$$

where $\mathbf{x}$ denotes the design variables, i.e., the solution; T is a user-specified parameter stating for how long a solution is used; $\boldsymbol{\alpha}$ represents the environmental state that specifies the fitness function f. The environment state at time i is denoted as $\boldsymbol{\alpha}(i)$.

To account for the second aforementioned situation where it is desirable to use a solution for as much time as possible, as long as the solution maintains its fitness above a predefined threshold,[2] we proposed the following robustness definition to quantify the amount of time a solution can maintain its fitness above a user-specified threshold [20]. Assuming a solution $\mathbf{x}$ is determined at time t, we have the corresponding robustness

$$F^s(\mathbf{x}, t, V) = \begin{cases} 0, & \text{if } f(\mathbf{x}, \boldsymbol{\alpha}(t)) < V \\ \max\{l | t \le i \le t + l : f(\mathbf{x}, \boldsymbol{\alpha}(i)) \ge V\}, & \text{else} \end{cases} \tag{2}$$

where V is a user-specified parameter of the fitness threshold.

Assuming that the dynamic environment can be represented as a sequence of static fitness functions $(f_1, f_2, \ldots, f_N)$ during a considered time interval $[t_0, t_{\text{end}})$, (1) and (2) are reduced to the discrete forms, respectively, in

$$F^a(\mathbf{x}, t, T) = \frac{1}{T}\sum_{i=0}^{T-1} f_{t+i}(\mathbf{x}). \tag{3}$$

$$F^s(\mathbf{x}, t, V) = \begin{cases} 0, & \text{if } f_t(\mathbf{x}) < V \\ 1 + \max\{l | \forall i \in \{t, t+1, \ldots, t+l\} : f_i(\mathbf{x}) \ge V\}, & \text{else} \end{cases} \tag{4}$$

where the fitness function f in either (3) or (4) belongs to the sequence $(f_1, f_2, \ldots, f_N)$. The two definitions in (3) and (4) are termed "average fitness" and "survival time" in [20], and we follow this convention in this paper.

Assuming that the dynamic environment can be represented as a sequence of static fitness functions $(f_1, f_2, \ldots, f_N)$ during a considered time interval $[t_0, t_{\text{end}})$, the ROOT problem is defined as

$$\begin{aligned} &\max \sum_{i=1}^{N} F(\mathbf{x}_i) \\ &\text{s.t. } \mathbf{x}_i \textit{ is} \text{ feasible}, \ 1 \le i \le N \end{aligned} \tag{5}$$

where $\mathbf{x}_i$ denotes the robust solution determined at time step i when the dynamic environment is represented as f_i. $F(\mathbf{x}_i)$ is the robustness of solution $\mathbf{x}_i$, and in this paper F is either average fitness or survival time.[3] It should be noted that any ROOT method is supposed to solve the ROOT problem in an on-line manner. In other words, a ROOT method starts when the dynamic environment takes the form of f_1 and determines a solution $\mathbf{x}_1$ within a computational budget Δe. Then, the ROOT method will determine a solution $\mathbf{x}_2$ within the computational budget Δe after the environment changes from f_1 to f_2. The process repeats till the last static fitness function f_N. Also, the computational budget Δe should be generally smaller than the time interval between two successive environmental changes.

At the current time step t, the objective of ROOT in this paper is to find a solution that maximizes average fitness or survival time. In contrast, for TMO problems, the objective at the current time step t is to find a solution that maximizes the fitness function f_t. Hence, the way the decision maker determines and implements solutions to ROOT problems is different from that to TMO problems. A solution implemented at a certain time step in ROOT can be used for multiple consecutive time steps (i.e., it is not necessary to implement a new solution every time the environment changes as what is

[2]Only maximization problems are considered in this paper without loss of generality.

[3]This means we have two versions of ROOT: one is about maximizing average fitness, and the other is about maximizing survival time.

assumed in TMO), due to the definition of average fitness or survival time. Moreover, it should be noted that for both robustness definitions in (3) and (4), at the current time step t, only f_t can be exactly evaluated, and a solution's fitness at future time steps (i.e., time step $t+1$, $t+2$, ...) has to be predicted either explicitly or implicitly.

In order to evaluate algorithms for ROOT, appropriate benchmarks are needed. Over the years, researchers have developed various DOP benchmarks to test an algorithm's TMO ability. Since existing DOP benchmarks are essentially benchmarks that define the dynamic fitness functions (DFFs) (i.e., a sequence of static fitness functions in the discrete-time case) for DOPs, those DOP benchmarks could be potentially used for testing an algorithm's ROOT ability as well.

There are mainly three categories in existing DOP/DFF benchmarks. In the first category, the DFF switches among several fixed fitness functions, e.g., the oscillating fitness landscape [21] and the dynamic knapsack problem [22].

The second category includes benchmarks that are built by constructing baseline fitness functions with configurable parameters and developing dynamics to change those configurable parameters. Typical examples are the moving peaks benchmark [23], DF1 dynamic benchmark [24], the multiobjective dynamic test problem generator [25], the dynamic rotation peak benchmark together with the dynamic composition benchmark [26], and the dynamic constrained benchmark [5].

Compared to the first two categories, in which the DFF changes over time, the third category involves benchmarks where the DFF stays unchanged but a solution $\mathbf{x}$ has to go through a transformation before being evaluated, and the transformation rule is subject to environmental changes. A representative example is the exclusive-or generator for binary encoded problems [27] and continuous domains [28].

In contrast to the fitness-landscape-oriented DOP benchmarks in the aforementioned three categories, another seminal work was developed in [10], where the authors proposed a DOP benchmark based on general characteristics of real-world DOPs.

Given so many DOP benchmarks available, it turns out that it is not sufficient to test an algorithm's ROOT ability on existing DOP benchmarks. The reason is that it is generally difficult to know the absolute best average fitness for existing DOP benchmarks given a time window T ($T > 2$). Also, it is generally difficult to know the absolute best survival time for existing DOP benchmarks given a fitness threshold V. The reasons for both difficulties will be explained, respectively, in later sections of this paper. The lack of knowledge about the absolute best performance makes the evaluation and comparison of algorithm in terms of ROOT on existing benchmarks difficult if not impossible.

The rest of this paper is organized as follows. Having introduced two robustness definitions, i.e., average fitness and survival time, in ROOT, we explain in Section II the assumptions in ROOT so that people can understand potential applications of ROOT better. This is followed by discussing two primary aspects of difficulties in finding optimal solutions to ROOT in terms of average fitness or survival time. In Section III, we explain in detail why it is so hard to obtain the absolute best average fitness and the absolute best survival time on existing DOP benchmarks. In Section IV, we develop two new benchmarks by defining two new baseline fitness functions for maximizing average fitness and maximizing survival time, respectively. Both baseline fitness functions are specially designed, which are scalable to any number of dimensions and computationally efficient. Most importantly, both baseline fitness functions allow the exact calculation of the absolute best average fitness and the absolute best survival time, respectively. The calculations are based on the corresponding theorems proved in this paper. With the absolute best average fitness and the absolute best survival time available in the two proposed benchmarks, respectively, we test two state-of-the-art ROOT methods and two representative TMO methods on the two proposed ROOT benchmarks in Section V. Finally, the conclusion is given in Section VI.

II. ROOT

The objective of ROOT is different from that of TMO, where a solution that maximizes the fitness function f_t is sought at time step t. However, it is easy to understand that TMO does not necessarily contradict with ROOT as solutions that maximize f_t might also maximize average fitness or survival time depending on how the DFF changes over time.

It is important that implicit assumptions made explicit about ROOT, so people can understand the potential applications of ROOT. In the following, we make the least number of assumptions about ROOT in the hope that ROOT can be as applicable as possible.

Firstly, we assume that the dynamic environment, i.e., the DFF, changes suddenly with constant status between two successive changes. This means that the DFF can be represented as a sequence of static fitness functions during a considered time interval $[t_0, t_{\text{end}})$. The primary reason to assume that the dynamic environment changes discretely is as follows. It is natural to divide those types of DOPs where implementing a solution involves human operation or huge cost into time windows, during each of which the environment is considered static, according to the comprehensive survey of real-world DOPs in [29, Ch. 3]. Also, detecting environmental changes is considered as a separate task from ROOT.

Secondly, the task of the decision maker in ROOT is to determine a solution within a computational budget when an environmental change occurs. The decision maker would then decide whether to implement the determined solution for the new environment. In the context of average fitness, the determined solution will not be implemented until the previously implemented solution has been used for T time steps. In the context of survival time, the determined solution will not be implemented as long as the previously implemented solution maintains its fitness above the predefined fitness threshold V.

Thirdly, at each time step when the DFF is considered static, the decision maker should be able to evaluate a number of solutions and obtain their fitnesses for the static fitness function at that time step. In other words, ROOT can deal with black-box DFFs, which means that the analytical form of DFF can be unavailable and the only requirement is that the

decision maker can evaluate a solution's fitness at the current time step.

Fourthly, a solution's future fitness[4] should be predictable to some extent. In ROOT, at the current time step, we would like to find a solution that maximizes average fitness or survival time, both of which involve a solution's future fitness. As we are unable to evaluate a solution's future fitness at the current time step, certain prediction techniques are needed, which will be trained based on historical evaluation information. The historical evaluation information consists of a set of evaluated solutions with their fitnesses at previous time steps, which is essentially a set of triplets in the form $(\mathbf{x}, f_i(\mathbf{x}), i)$. $(\mathbf{x}, f_i(\mathbf{x}), i)$ means that solution $\mathbf{x}$ has been evaluated at previous time step i with its fitness for f_i being $f_i(\mathbf{x})$. Only under the assumption about predictability in solution's future fitness, information gathered in the past is useful to guide the search in the present in ROOT. Otherwise, the environment is said to change completely randomly and there is no need to use historical evaluation information.

Finally, solutions implemented in ROOT are used for a long time period, during which the environment changes. Therefore, it is required that the dimensionality of design variables does not change over time. However, it should be noted that we can have the constraints for design variables changing over time in ROOT, although in this paper we only investigate the case where the solution space[5] $\mathcal{S}$ stays constant over time. Also, in this paper, we do not consider the issue of time-linkage [30], which means that maximizing the sum in (5) is equivalent to maximizing the corresponding robustness respectively at each time step.

From the discussion above, we can see that at the current time step, in order to find optimal robust solutions in terms of average fitness or survival time, a solution's future fitness needs to be predicted. This imposes two major difficulties in ROOT problems. The first one would be in building learning models for the prediction task. Given historical evaluation information, under our assumptions in ROOT, it is not straightforward as to how to build such a learning model that can be used to predict a solution's future fitness. The second major difficulty lies in that there is no guarantee that a solution's future fitness can be predicted perfectly. In other words, when we apply EAs to ROOT problems, solutions are evolved inevitably based on inaccurate average fitness or survival time. In the following, we discuss in detail difficulties encountered in ROOT from both perspectives.

A. Difficulties in Predicting Solution's Future Fitness in ROOT

The most intuitive way to predict a solution's future fitness would be formulating it as a time series prediction problem and then employing time series prediction models, e.g., the autoregressive model [31], to predict a solution's future fitness. However, should the time series data be a solution's past fitnesses or models of past fitness functions does not have a confirm answer. Moreover, how to generate the time series data is not straightforward. For instance, if we decide the time series data is a series of a solution's past fitnesses, it may happen that a solution has not been evaluated at certain previous time steps, and therefore the solution's fitness at some previous time steps cannot be fetched directly from the historical evaluation information. Finally, given the time series data, the time series prediction task is itself a hard problem [32].

B. Difficulties in Evolving Population Based on Inaccurate Information in ROOT

We are interested in applying EAs to ROOT problems. The metric that is used to differentiate good solutions from bad ones in EAs plays an important role in optimization process as it guides the population to converge to good solutions. For TMO problems, a solution's future fitness is not considered, and the corresponding metric is a solution's current fitness. In contrast, solutions to ROOT problems are compared based on a solution's estimated/predicted average fitness or survival time. Since a solution's future fitness cannot be exactly evaluated, the estimated/predicted average fitness or survival time is inevitably inaccurate or noisy. Evolving solutions based on an inaccurate or noisy metric has long been studied in evolutionary optimization with noisy fitness functions [2] and surrogate-assisted evolutionary optimization [33]. A major concern with an inaccurate or noisy metric is that it may lead EAs to converge to a false optimum. This concern applies to EAs for ROOT problems as well.

III. Difficulties in Calculating the Absolute Best Average Fitness and the Absolute Best Survival Time

Existing DOP benchmarks are essentially DFF benchmarks. Suppose a sequence of static fitness functions of length N, $(f_1, f_2, \ldots, f_N)$, has been generated by a DFF benchmark. According to the average fitness definition in (3), the absolute best average fitness starting from f_t with time window T, $t + T \leq N$, is

$$\max \left\{ F^a(\mathbf{x}, t, T) | \mathbf{x} \in \mathcal{S} \right\} \tag{6}$$

where $\mathcal{S}$ denotes the solution space of $\mathbf{x}$. According to the survival time definition in (4), the absolute best survival time starting from f_t with the fitness threshold V is

$$\max \left\{ F^s(\mathbf{x}, t, V) | \mathbf{x} \in \mathcal{S} \right\}. \tag{7}$$

A. Difficulties in Calculating the Absolute Best Average Fitness

The difficulty to obtain the absolute best average fitness for existing DFF benchmarks lies in the fact that usually the function form of $F^a(\mathbf{x}, t, T)$ when $T \geq 2$ cannot be reduced to the same form as the corresponding baseline fitness function. Therefore, the procedure that is used to obtain an optimum of the corresponding baseline fitness function cannot be used to obtain an optimum of the average fitness function in (3).

Taking the widely used moving peaks benchmark [23] and the DF1 dynamic benchmark [24] for example, the baseline

[4]If the current time step is t, a solution's future fitness means its fitness at future time step $t + i$, $i \geq 1$.

[5]The solution space is a set of all candidate solutions.

fitness function in both benchmarks takes a similar form as

$$f_t(\mathbf{x}) = \max_{i=1}^{i=m} \left\{ h_t^i - w_t^i * \left|\left|\mathbf{x} - \mathbf{c}_t^i\right|\right|_2 \right\} \tag{8}$$

where scalars h_t^i, w_t^i, and vector $\mathbf{c}_t^i$, $\mathbf{c}_t^i = (c_1^i, \ldots, c_d^i)$, denote the height, the width and the center of the ith peak function at time step t; $\mathbf{x}$ is the vector of design variables; m is the total number of peaks, and d is the number of dimensions. According to the function form in (8) and supposing the solution space $\mathcal{S}$ being the d-dimensional vector space over the real numbers $\mathcal{R}^d$, it is easy to infer that an optimal solution for the baseline fitness function in (8) is $\mathbf{c}_t^{j^*}$ with its fitness taking the value $h_t^{j^*}$ where $j^* = \arg\max_i h_t^i$, $1 \le i \le m$. When we use the baseline fitness function in (8) to generate the DFF, the function form of average fitness, $F^a(\mathbf{x}, t, T)$, becomes (supposing $T = 2$)

$$\frac{1}{2}\left(\max_{i=1}^{i=m} \left\{ h_t^i - w_t^i * \left|\left|\mathbf{x} - \mathbf{c}_t^i\right|\right|_2 \right\} + \max_{i=1}^{i=m} \left\{ h_{t+1}^i - w_{t+1}^i * \left|\left|\mathbf{x} - \mathbf{c}_{t+1}^i\right|\right|_2 \right\} \right). \tag{9}$$

It is easy to verify that we are now unable to use the same inference technique to obtain the absolute best average fitness ($T \ge 2$). The reason is that the function form of average fitness in (9) cannot be reduced to the same form as the baseline fitness function in (8).

A natural attempt to find the absolute best average fitness for DFFs generated using the baseline fitness function in (8) would be to set the derivative of the function of average fitness to zero and then obtain the stationary points. However, the analytical solutions of stationary points are not available because of the function form in (9).

Although we illustrate the difficulty in calculating the absolute best average fitness using two existing DFF benchmarks, we argue that for most existing DFF benchmarks [23], [24], [26], the same difficulty in calculating the absolute best average fitness applies. However, there exists one DFF benchmark [25] where the function form of average fitness can be reduced to the form of the corresponding baseline fitness function, and therefore it is capable to calculate the absolute best average fitness just as calculating the optimal fitness for the baseline fitness function in [25]. Nonetheless, we would like to contribute a new benchmark for maximizing average fitness in ROOT. This is partly because, for the DFF benchmark in [25], all possible optimal solutions of average fitness are confined within a fixed Pareto set. The reason is that the DFF is generated in [25] by aggregating different objects of a fixed multiobjective optimization problem and changing the aggregating weights.

B. Difficulties in Calculating the Absolute Best Survival Time

The difficulty in calculating the absolute best survival time in (7) originates from the difficulty associated with general nonlinear arithmetic constraint satisfaction problems (CSPs) [34]. Given a sequence of static fitness functions $(f_1, f_2, \ldots, f_N)$, the problem of calculating the absolute best survival time starting from f_t is equivalent to the problem of finding the maximal number that the variable l can take for which there exists a solution $\mathbf{x}^*$, $\mathbf{x}^* \in \mathcal{S}$, satisfying the following array of arithmetic constraints simultaneously:

$$\begin{cases} f_t(\mathbf{x}^*) \ge V \\ f_{t+1}(\mathbf{x}^*) \ge V \\ \vdots \\ f_{t+l}(\mathbf{x}^*) \ge V. \end{cases} \tag{10}$$

The problem of answering whether there exists a solution $\mathbf{x}^* \in \mathcal{S}$ that satisfies all the constraints in (10) for a fixed l, $l \ge 0$, is one type of CSPs. Therefore, being able to solve the corresponding CSP is a necessary condition of calculating the absolute best survival time. Supposing we can solve the corresponding CSP successfully, we can obtain the absolute best survival time, starting from any fitness function f_t, $1 \le t \le N$, in the sequence $(f_1, f_2, \ldots, f_N)$, by separately solving a number of corresponding CSPs with l being $1, 2, \ldots, N - t$.

As stated in [34], solving arbitrary nonlinear arithmetic CSPs over the real numbers is undecidable. This means if f_{t+i}, $0 \le i \le l$, is an arbitrary nonlinear function of $\mathbf{x}$, it is impossible to construct a single algorithm that will always lead to a yes/no answer as to whether there exists a solution $\mathbf{x}^*$ simultaneously satisfying all the constraints in (10). If f_{t+i}, $0 \le i \le l$, is a linear function of $\mathbf{x}$, the corresponding CSP in (10) is reduced to a linear programming problem and can be solved in polynomial time. Also, if both the baseline fitness function f_{t+i}, $0 \le i \le l$, and the solution space $\mathcal{S}$ are convex, the corresponding CSP can be solved satisfactorily [35].

Based on the above discussion, we can see that in order to employ a general CSP solver to solve the CSP with a fixed l in (10), the baseline fitness function f_t is required to be either linear or convex together with a convex solution space $\mathcal{S}$. Alternatively, we can require the baseline fitness function to take a specific form (more complexed than linear and convex) based on which we can calculate the absolute best survival time. We take the latter way in this paper to develop a new baseline fitness function (i.e., a new baseline fitness function for a new benchmark) for the study of maximizing survival time in ROOT. The reason is that the requirement of the baseline fitness function being linear or convex is too strong that may only represent a small portion of real-world problems.

IV. Tunable Benchmark Problems for ROOT

In this section, we describe in detail the construction of two benchmark problems for ROOT. The first benchmark is used specially for maximizing average fitness in ROOT, and the other is used specially for maximizing survival time in ROOT. It is worth noting that, although we use loosely the term of "two benchmark problems," it actually includes many different benchmark instances given different dynamics for the corresponding baseline fitness function. Two here should be regarded as two types of baseline fitness functions.

The two benchmarks are developed by first developing respectively two baseline fitness functions with configurable

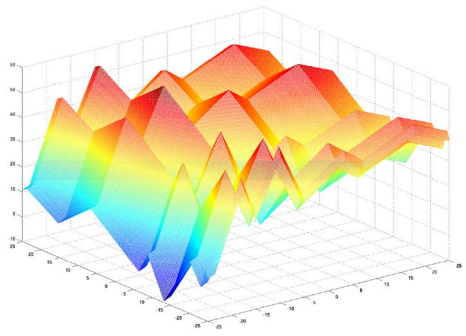


Fig. 1. One fitness landscape example with five peaks along each dimension generated by the baseline fitness function in RMPB-I.

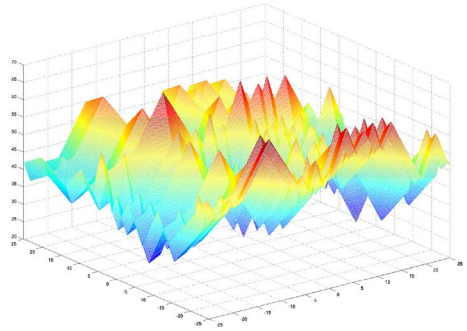


Fig. 2. One fitness landscape example with 20 peaks along each dimension generated by the baseline fitness function in RMPB-I.

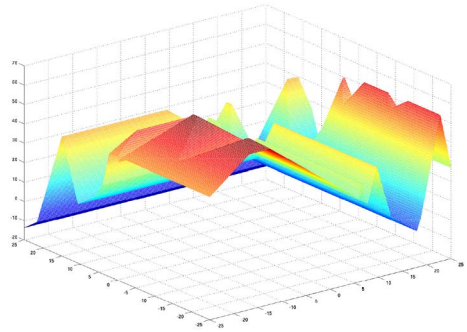


Fig. 3. One fitness landscape example with five peaks along each dimension generated by the baseline fitness function in RMPB-II.

parameters. After that, we suggest some desired dynamics that are used to change the configurable parameters in our baseline fitness functions to produce DFFs. We prove two lemmas and one theorem based on which the absolute best average fitness is calculated for DFFs generated from the first baseline fitness function. Another two lemmas and one theorem are proved with regard to calculating the absolute best survival time for DFFs generated from the second baseline fitness function.

Since the two proposed ROOT benchmarks have the "peak" characteristic as defined in the moving peaks benchmark [23] and aim at testing an algorithm's ROOT ability (in terms of maximizing average fitness and survival time, respectively), we term the two benchmarks robust moving peaks benchmark (RMPB). The benchmark for maximizing average fitness is denoted as RMPB-I, and the benchmark for maximizing survival time is denoted as RMPB-II.

A. Baseline Fitness Functions

The baseline fitness function in RMPB-I for maximizing average fitness in ROOT is

$$f_t^a(\mathbf{x}) = \frac{1}{d}\sum_{j=1}^{d}\max_{i=1}^{i=m}\left\{h_t^{ij} - w_t^{ij} * \left|x_j - c_t^{ij}\right|\right\} \tag{11}$$

and the baseline fitness function in RMPB-II for maximizing survival time in ROOT is

$$f_t^s(\mathbf{x}) = \min_{j=1}^{j=d}\left\{\max_{i=1}^{i=m}\left\{h_t^{ij} - w_t^{ij} * \left|x_j - c_t^{ij}\right|\right\}\right\} \tag{12}$$

where, in both baseline fitness functions, scalars h_t^{ij}, w_t^{ij}, and c_t^{ij} denote the height, the width, and the center of the ith peak function for the jth dimension at time step t. The ith peak function for the jth dimension in both baseline fitness functions takes the same form: $h_t^{ij} - w_t^{ij} * |x_j - c_t^{ij}|$. The only difference between f_t^a and f_t^s is that f_t^a is the average over all dimensions while f_t^s takes the minimal value among all dimensions. $\mathbf{x}$ represents the design variables with d dimensions ($\mathbf{x} = (x_1, \ldots, x_d)$), and m is the number of peaks along each dimension. Without loss of generality, we set the solution space S as the constrained d-dimensional vector space $[a, b]^d$ where a and b are two real numbers. Also, we require that each w_t^{ij} takes positive values and each c_t^{ij} belongs to the interval $[a, b]$.

The construction of the two baseline fitness functions is motivated as follows. On the one hand, we would like both baseline fitness functions to be multimodal, scalable to any number of dimensions, and computationally efficient. More importantly on the other hand, we should be able to calculate the absolute best average fitness in DFFs generated using the first baseline fitness function and the absolute best survival time in DFFs generated using the second baseline fitness function. However, we would like to mention that the two baseline fitness functions are developed not to represent any specific real-world situation but to provide proper platforms for the study of ROOT problems.

We generate two examples of fitness landscape using the baseline fitness function in (11) in Figs. 1 and 2. We generate another two examples of fitness landscape using the baseline fitness function in (12) in Figs. 3 and 4. In Figs. 1 and 3, we have 5 peaks along each dimension (2-D solution space), and we generate the landscapes randomly with each height ranging from 30 to 70, each width ranging from 1 to 13, and each center ranging from -25 to 25. The same rule is applied to Figs. 2 and 4 except that we set m, i.e., the number of peaks along each dimension, to 20. We can see that the fitness landscape in Fig. 2 is more rugged than that in Fig. 1, and so is the fitness landscape in Fig. 4 than that in Fig. 3.

B. Dynamics for the Baseline Fitness Functions

We do not restrict the types of dynamics that are applied to the configure parameters, i.e., height h_t^{ij}, width w_t^{ij}, and center c_t^{ij}, in the two baseline fitness functions to generate DFFs. In other words, users can define their own dynamics for their study of ROOT problems using the two baseline

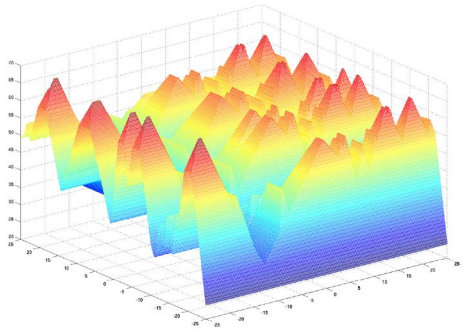


Fig. 4. One fitness landscape example with 20 peaks along each dimension generated by the baseline fitness function in RMPB-II.

fitness functions. However, we suggest the following 6 types of dynamics that have been developed in CEC09 dynamic optimization competition [26]. The six different types of dynamics cover common dynamics found in real-world DFFs, which are described as follows.

1) *Small Step*

$$\Delta\phi = \gamma * ||\phi|| * r * \phi_{\text{severity}}. \tag{13}$$

2) *Large Step*

$$\Delta\phi = ||\phi|| * (\gamma * \text{sign}(r) + (\gamma_{\max} - \gamma) * r) * \phi_{\text{severity}}. \tag{14}$$

3) *Random*

$$\Delta\phi = \mathcal{N}(0, 1) * \phi_{\text{severity}}. \tag{15}$$

4) *Chaotic*

$$\phi_{t+1} = \phi_{\min} + A * (\phi_t - \phi_{\min}) * (1 - (\phi_t - \phi_{\min})/||\phi||). \tag{16}$$

5) *Recurrent*

$$\phi_t = \phi_{\min} + ||\phi|| * \left(\sin\left(\frac{2\pi}{P}t + \varphi\right) + 1\right)/2. \tag{17}$$

6) *Recurrent With Noise*

$$\phi_t = \phi_{\min} + ||\phi|| * \left(\sin\left(\frac{2\pi}{P}t + \varphi\right) + 1\right)/2 + \mathcal{N}(0, 1) * \text{noise}_{\text{severity}} \tag{18}$$

where ϕ represents a configurable parameter in DFFs. ϕ_t is the value of ϕ at time step t, and $\Delta\phi$ denotes the change in ϕ between two consecutive time steps: $\phi_{t+1} = \phi_t + \Delta\phi$. $\phi_{\min}$, $||\phi||$, and ϕ_{severity} denote the minimum value of ϕ, the range of ϕ, and the change severity of ϕ, respectively. ϕ_{severity} basically controls the magnitude of the change in ϕ between two consecutive time steps. γ and $\gamma_{\max}$ are constant values, which are set to 0.04 and 0.1, respectively. A logistic function is used for the chaotic change: A is a positive constant in the interval $(1, 4)$. P is the period for the recurrent change and recurrent with noise change, and φ is the initial phase. r is a random number drawn uniformly from the interval $(-1, 1)$. $\text{sign}(r)$ returns 1 when r is positive, -1 when r is negative, and 0 otherwise. $\mathcal{N}(0, 1)$ is a random number drawn from the Gaussian distribution with mean 0 and variance 1. $\text{noise}_{\text{severity}}$ is the noise severity applied to the recurrent with noise change.

The height h_t^{ij} and width w_t^{ij} in (11) and (12) are updated using up-mentioned six dynamics. An additional technique using rotation matrix is employed to rotate the centers (vector $\mathbf{c}_t^i = (c_t^{i1}, c_t^{i2}, \ldots, c_t^{id})$ in (11) and (12)). More specifically, according to [26], the following algorithm is used to change the center.

1) l_r (l_r is even) number of dimensions are randomly selected from the d dimensions resulting a vector $(d_1, \ldots, d_{l_r})$.
2) For each pair of dimension d_i and dimension d_{i+1} $(1 \le i \le l_r - 1)$, construct a rotation matrix $\mathbf{R}_t(d_i, d_{i+1})$ that rotates the vector of centers in the plane $d_i - d_{i+1}$ by an angle θ_t from the d_ith axis to the d_{i+1}th axis.
3) Since rotation matrices are orthogonal, an overall rotation matrix $\mathbf{R}_t$ is obtained via: $\mathbf{R}_t = \mathbf{R}_t(d_1, d_2) * \cdots * \mathbf{R}_t(d_{l_r-1}, d_{l_r})$.
4) The new vector of centers is produced by setting $\mathbf{c}_{t+1}^i = \mathbf{c}_t^i * \mathbf{R}_t$.

Moreover, the rotation angle θ_t is subject to the six different dynamics. Therefore, every time the DFF changes, the rotation angle θ_t is updated first, and the new angle is used to construct the rotation matrix that eventually changes the position of each center.

Additionally, we introduce a parameter Δe for both RMPB-I and RMPB-II. Δe is measured in the number of fitness evaluations and is used to measure the computational budget for the decision maker to determine a solution right after an environmental change in ROOT problems. It should be noted that Δe should be generally smaller than the frequency of environmental changes measured in the number of fitness evaluations. The reason to introduce Δe, instead of a parameter that controls how frequently the DFF changes, is that usually a solution has to be found before a certain deadline in many real-world DOPs (Δe is called the deadline parameter in [29, Ch. 3, p. 67]). The deadline is usually before the next possible environmental change. In other words, solutions usually have to be implemented before the next environmental change happens.

C. Relationship Between the Benchmark Parameters and ROOT Problem Difficulties

We can tune the difficulties of ROOT problems in the following aspects. The complexity of the baseline fitness functions can be varied by changing the number of peaks, m, along each dimension and the number of dimensions. The larger m is, the more rugged the fitness landscape is, and the more difficult it is for EAs to find a good solution to ROOT problems. In addition, we can vary ROOT problem difficulties by tuning the parameter Δe as Δe controls the number of fitness evaluations for EAs to determine a solution at a time step. Also, different dynamics in the DFF should have an influence on the performance of methods for ROOT problems.

D. Calculating the Absolute Best Average Fitness in RMPB-I

In this subsection, we prove two lemmas and one theorem for the purpose of calculating the absolute best average fitness in (6) for DFFs generated in RMPB-I.

Suppose a sequence of fitness functions $(f_1^a, f_2^a, f_3^a, \ldots, f_N^a)$ has been generated in RMPB-I. Without loss of generality, we would like to calculate the following absolute best average fitness: $\max\{F^a(\mathbf{x}, t, T)|\mathbf{x} \in \mathcal{S}\}$, which starts from f_t^a in the sequence $(f_1^a, f_2^a, f_3^a, \ldots, f_N^a)$ ($t < N$, $t + T - 1 \leq N$, and $T \geq 2$). $\max\{F^a(\mathbf{x}, t, T)|\mathbf{x} \in \mathcal{S}\}$ takes the form as

$$\max\left\{F^a(\mathbf{x}, t, T)|\mathbf{x} \in \mathcal{S}\right\} = \max\left\{\frac{1}{T}\sum_{i=0}^{T-1} f_{t+i}^a(\mathbf{x})|\mathbf{x} \in \mathcal{S}\right\}. \quad (19)$$

Firstly, we define the maximal average fitness (MAF) of a number of fitness functions $(g_1, g_2, \ldots, g_k)$ as

$$\text{MAF}(g_1, g_2, \ldots, g_k) = \max\left\{\frac{1}{k}\sum_{i=1}^{k} g_i(\mathbf{x})|\mathbf{x} \in \mathcal{S}\right\} \quad (20)$$

where g_i, $1 \leq i \leq k$, represents a fitness function of $\mathbf{x}$. All the fitness functions, i.e., g_i, $1 \leq i \leq k$, share the same solution space $\mathcal{S}$. By definition, $\max\{F^a(\mathbf{x}, t, T)|\mathbf{x} \in \mathcal{S}\} = \text{MAF}(f_t^a, f_{t+1}^a, \ldots, f_{t+T-1}^a)$. We use peak_t^{ij} to denote the ith peak function for the jth dimension at time step t in (11): $\text{peak}_t^{ij} = h_t^{ij} - w_t^{ij} * |x_j - c_t^{ij}|$, $a \leq x_j \leq b$. We have the following lemma.

Lemma 1: The MAF of a set of peak functions $\{\text{peak}_{t+k}^{i_k j}|0 \leq k \leq T-1\}$, i.e., $\text{MAF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+T-1}^{i_{T-1} j})$, can be achieved when x_j takes the value of one of the centers $c_{t+k}^{i_k j}$, $0 \leq k \leq T-1$.

Lemma 1 is proved in the Appendix. From Lemma 1, we can see that the MAF for a set of peak functions $\{\text{peak}_{t+k}^{i_k j}|0 \leq k \leq T-1\}$ equals $\max\{\frac{1}{T}\sum_{k=0}^{T-1}(h_{t+k}^{i_k j} - w_{t+k}^{i_k j} * |x_j - c_{t+k}^{i_k j}|) \mid x_j \in \{c_{t+k}^{i_k j}|0 \leq k \leq T-1\}\}$, which can be calculated by enumerating all the centers.

We use dim_t^j to denote the jth dimensional function at time step t in (11): $\text{dim}_t^j = \max_{i=1}^{i=m}\{h_t^{ij} - w_t^{ij} * |x_j - c_t^{ij}|\}$, $a \leq x_j \leq b$. We have the following lemma, which is proved in the Appendix.

Lemma 2: The MAF of a set of dimensional functions $\{\text{dim}_{t+k}^j|0 \leq k \leq T-1\}$, i.e., $\text{MAF}(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+T-1}^j)$, is equal to $\max\{\text{MAF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+T-1}^{i_{T-1} j}) \mid 1 \leq i_k \leq m, 0 \leq k \leq T-1\}$.

Theorem 1: The MAF of a set of fitness functions $\{f_{t+k}^a|0 \leq k \leq T-1\}$ is equal to $1/d\sum_{j=1}^{d}\max\{\text{MAF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+T-1}^{i_{T-1} j}) \mid 1 \leq i_k \leq m, 0 \leq k \leq T-1\}$.

Theorem 1 is proved based on Lemma 2 in the Appendix. Based on Lemma 1 and Theorem 1, we can exactly calculate $\text{MAF}(f_t^a, f_{t+1}^a, \ldots, f_{t+T-1}^a)$, which is the absolute best average fitness, i.e., $\max\{F^a(\mathbf{x}, t, T)|\mathbf{x} \in \mathcal{S}\}$, given a sequence of fitness functions $(f_1^a, f_2^a, \ldots, f_N^a)$ generated in RMPB-I.

E. Calculating the Absolute Best Survival Time in RMPB-II

In this subsection, we prove two lemmas and one theorem for the purpose of calculating the absolute best survival time in (7) for DFFs generated in RMPB-II.

Suppose a sequence of fitness functions $(f_1^s, f_2^s, f_3^s, \ldots, f_N^s)$ has been generated in RMPB-II. Without loss of generality, we would like to calculate the following absolute best survival time: $\max\{F^s(\mathbf{x}, t, V)|\mathbf{x} \in \mathcal{S}\}$, which starts from f_t^s ($1 \leq t \leq N$) in the sequence $(f_1^s, f_2^s, f_3^s, \ldots, f_N^s)$ with the fitness threshold V.

Firstly, we define the maximal intersection fitness (MIF) of a number of fitness functions $(g_1, g_2, \ldots, g_k)$ as

$$\text{MIF}(g_1, g_2, \ldots, g_k) = \max\left\{\min_{i=1}^{i=k} g_i(\mathbf{x})|\mathbf{x} \in \mathcal{S}\right\} \quad (21)$$

where g_i, $1 \leq i \leq k$, represents a fitness function of $\mathbf{x}$, and all the fitness functions, i.e., g_i, $1 \leq i \leq k$, share the same solution space $\mathcal{S}$. We use peak_t^{ij} to denote the ith peak function for the jth dimension at time step t in (12): $\text{peak}_t^{ij} = h_t^{ij} - w_t^{ij} * |x_j - c_t^{ij}|$, $a \leq x_j \leq b$. It is easy to verify that the MIF of $\text{peak}_t^{i_0 j}$ and $\text{peak}_{t+1}^{i_1 j}$, i.e., $\text{MIF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j})$, is equal to

$$\begin{cases} h_t^{i_0 j}, & \text{if } h_{t+1}^{i_1 j} - w_{t+1}^{i_1 j} * |c_t^{i_0 j} - c_{t+1}^{i_1 j}| \geq h_t^{i_0 j} \\ h_{t+1}^{i_1 j}, & \text{elseif } h_t^{i_0 j} - w_t^{i_0 j} * |c_{t+1}^{i_1 j} - c_t^{i_0 j}| \geq h_{t+1}^{i_1 j} \\ \frac{w_t^{i_0 j} * h_{t+1}^{i_1 j} + w_{t+1}^{i_1 j} * h_t^{i_0 j} - w_t^{i_0 j} * w_{t+1}^{i_1 j} * |c_t^{i_0 j} - c_{t+1}^{i_1 j}|}{w_t^{i_0 j} + w_{t+1}^{i_1 j}}, & \text{else.} \end{cases} \quad (22)$$

For a number of peak functions, we have the following lemma, which is proved in the Appendix.

Lemma 3: The MIF of a set of peak functions $\{\text{peak}_{t+k}^{i_k j}|0 \leq k \leq L-1\}$ ($L \geq 2$), i.e., $\text{MIF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+L-1}^{i_{L-1} j})$, is equal to $\min\{\text{MIF}(\text{peak}_{t+p}^{i_p j}, \text{peak}_{t+q}^{i_q j})|0 \leq p < q \leq L-1\}$.

We use dim_t^j to denote the jth dimensional function at time step t in (12): $\text{dim}_t^j = \max_{i=1}^{i=m}\{h_t^{ij} - w_t^{ij} * |x_j - c_t^{ij}|\}$, $a \leq x_j \leq b$. We have the following lemma, which is proved in the Appendix.

Lemma 4: The MIF of a set of dimensional functions $\{\text{dim}_{t+k}^j|0 \leq k \leq L-1\}$ ($L \geq 2$), i.e., $\text{MIF}(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+L-1}^j)$, is equal to $\max\{\text{MIF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+L-1}^{i_{L-1} j}) \mid 1 \leq i_k \leq m, 0 \leq k \leq L-1\}$.

Theorem 2: The MIF of a set of fitness functions $\{f_{t+k}^s|0 \leq k \leq L-1\}$ ($L \geq 2$), i.e., $\text{MIF}(f_t^s, f_{t+1}^s, \ldots, f_{t+L-1}^s)$, is equal to $\min_{j=1}^{d}\{\max\{\text{MIF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+L-1}^{i_{L-1} j}) \mid 1 \leq i_k \leq m, 0 \leq k \leq L-1\}\}$.

Theorem 2 is proved based on Lemma 4 in the Appendix. Based on (23), Lemma 3, and Theorem 2, we can exactly calculate $\text{MIF}(f_t^s, f_{t+1}^s, \ldots, f_{t+L-1}^s)$ for any L number of consecutive fitness functions in a sequence of fitness functions $(f_1^s, f_2^s, \ldots, f_N^s)$ generated in RMPB-II. The absolute best survival time, $\max\{F^s(\mathbf{x}, t, V)|\mathbf{x} \in \mathcal{S}\}$, starting from f_t^s, is equal to the largest number that L can take satisfying $\text{MIF}(f_t^s, f_{t+1}^s, \ldots, f_{t+L-1}^s) \geq V$. Therefore, the absolute best survival time, $\max\{F^s(\mathbf{x}, t, V)|\mathbf{x} \in \mathcal{S}\}$, starting from f_t^s, is equal to

$$\begin{cases} 0, & \text{if } \max\left\{f_t^s(\mathbf{x})|\mathbf{x} \in \mathcal{S}\right\} < V \\ \max\left\{L|\text{MIF}\left(f_t^s, f_{t+1}^s, \ldots, f_{t+L-1}^s\right) \geq V\right\}, & \text{else.} \end{cases} \quad (23)$$

V. Behavior of Existing Methods for ROOT

In this section, experimental studies are conducted regarding the performance of existing methods on RMPB-I and

TABLE I
PARAMETER SETTINGS OF THE TEST PROBLEMS

number of peaks m	5
number of dimensions d	2
search range $[a, b]$	$[-25, 25]$
height range	$[30, 70]$
width range	$[1, 13]$
angle range	$[-\pi, \pi]$
$height_{severity}$	5.0
$width_{severity}$	0.5
$angle_{severity}$	1.0
initial angle θ	0
γ	0.04
γ_{max}	0.1
chaotic constant A	3.67
period P	12
$noisy_{severity}$	0.8
number of dimensions l_r for rotation	2
time window T	2,6,10
fitness threshold V	40,45,50
computational budget at each time step Δe	2500

RMPB-II. The purpose is to investigate the strengths and weaknesses of different methods on ROOT problems with different dynamics.

A. Experimental Settings

1) Test Problems: We generate six benchmark instances in RMPB-I by applying the six types of dynamics defined in (13) and (18) to the first baseline fitness function in (11). In the meantime, we apply those six types of dynamics to the second baseline fitness function in (12) to produce six benchmark instances in RMPB-II. We refer each of them as TP_{ij}, $i = 1, 2$, $j = 1, 2, \ldots, 6$, where TP_{ij} means the benchmark instance that is generated by applying the jth dynamic to the ith baseline fitness function. TP_{1j}, $j = 1, 2, \ldots, 6$, are used for maximizing average fitness in ROOT, and TP_{2j}, $j = 1, 2, \ldots, 6$, are used for maximizing survival time in ROOT. For each TP_{ij} in this paper, we generate 200 consecutive fitness functions as follows. For the first fitness function, we randomly initialize the heights and the widths in their corresponding ranges, and the centers are randomly initialized across the solution space. To generate the next fitness function, we apply the jth dynamic to the current heights, widths, and rotation angle. The centers of the next fitness function are obtained by rotating the centers of the current fitness function using the updated rotation angle. Whenever the heights, widths, or centers get out of their corresponding ranges, we reset them to their up limits (if larger than up limits) or low limits (if lower than low limits). Note the following exceptions. For the chaotic change, centers are updated dimension by dimension using (16) rather than the rotation technique. For the recurrent change and the recurrent with noise change, θ is fixed to $2\pi/P$ where P is the period, and the centers are rotated following a fixed direction. A summary of all test problem parameters is presented in Table I.

2) Methods Investigated for ROOT Problems: We select four representative methods from the literature to test their ROOT abilities. The first approach is a simple particle swarm optimizer (PSO) with a restart strategy, which we will denote as "RPSO" hereafter. The restart strategy means that the best solution found in terms of the fitness for the last fitness function at the last time step is copied into the initial population for the current time step whenever the environment changes and that all other particles are initialized randomly. The second approach can be seen as an ideal TMO algorithm, in which each fitness function's optimum is chosen as the ROOT solution at each time step. The ideal TMO approach, denoted as "optimum" hereafter, can be viewed as the best any TMO approach can do in terms of TMO. The third and fourth approaches are two latest methods developed specially for ROOT problems. The third approach is Jin *et al.*'s framework [36], denoted as "Jin's" hereafter. A global radial basis function network (RBFN) is employed as the surrogate model to approximate a solution's previous fitness in Jin's. For predicting a solution's future fitness in Jin's, the autoregression (AR) [37] model with order 4 is employed. One estimated previous fitness and four predicted future fitnesses are used in the metric[6] in Jin's, the setting of which is reported to have the best performance in [36]. For more details of Jin's, readers are referred to [36]. The fourth approach is the method developed by Fu *et al.* [20], which we will denote as "Fu's" hereafter. The same RBFN and AR models are used in Fu's as in Jin's except that the metrics used for respectively maximizing average fitness and survival time in Fu's are different from the metric in Jin's. Furthermore, the control parameter in the metrics in Fu's is set to 0. For more details of Fu's, readers are referred to [20].

For methods RPSO, Jin's, and Fu's, they all employ the same constriction version of PSO [38] as the optimizer. The swarm population size is 50. The constants c_1 and c_2 that are used to bias a particle's attraction to the local best and the global best are both set to 2.05, and therefore the constriction factor χ takes the value 0.729844. The velocity of particles is constricted within the range $[-v_{\max}, v_{\max}]$. The value of $v_{\max}$ is set to the range of the search space, which is 50 in our case.

We believe it is necessary to test TMO methods for the purpose of ROOT. At the current time step, the objective of TMO is to find a solution maximizing the current fitness function. In contrast, the objective of ROOT is to find a solution that maximizes average fitness or survival time. It is easy to note that a solution's current fitness does not necessarily contradict with the solution's average fitness or survival time. In other words, whether a TMO method would be successful in finding robust solutions to ROOT problems largely depends on how the DFF changes over time, and the question is under what circumstances will TMO methods be effective or not in solving ROOT problems. The reason to employ RPSO and "optimum" methods is that they serve to represent a vast majority of approaches designed for the purpose of TMO. The ideal TMO approach, i.e., optimum approach, is selected in the hope that the effort to enumerate all state-of-the-art TMO approaches can be saved. Actually, optimum approach is the best any algorithm can do in terms of TMO. Besides, the reason to include Jin's and Fu's methods is that they are, to the best of our knowledge, the only two methods that have been designed specially for ROOT problems.

[6]The metric is a function that returns a real number for a candidate solution to quantify the quality of the solution.

TABLE II
MEAN PERFORMANCE IN (24) OVER 30 RUNS UNDER THE ROBUSTNESS AVERAGE FITNESS WITH DIFFERENT SETTINGS OF TIME WINDOW T

Test	RPSO			Optimum			Jin's			Fu's		
Problem	$T=2$	$T=6$	$T=10$	$T=2$	$T=6$	$T=10$	$T=2$	$T=6$	$T=10$	$T=2$	$T=6$	$T=10$
TP_{11}	51.65-	46.43-	45.12-	51.96-	46.26-	44.87-	51.50-	48.79	48.21	52.41	48.87	48.31
TP_{12}	44.90+	31.46-	27.90-	45.63+	30.97-	28.02-	40.18-	32.91	31.78	44.17	32.68	31.62
TP_{13}	48.88+	40.58+	38.51+	48.31+	38.97-	36.73-	43.60-	38.97-	37.48	47.28	39.25	37.34
TP_{14}	47.94-	37.41-	35.89-	48.89	37.59-	36.06-	46.10-	41.22	40.77-	48.92	41.50	41.28
TP_{15}	52.38+	34.21-	33.32-	52.58+	34.83-	34.33-	49.18-	36.64-	35.74-	51.33	37.45	38.82
TP_{16}	56.72+	44.40+	41.26	56.43+	44.46+	41.59	55.07-	43.35	40.66-	56.21	43.41	41.28

TABLE III
MEAN PERFORMANCE IN (24) OVER 30 RUNS UNDER THE ROBUSTNESS SURVIVAL TIME WITH DIFFERENT SETTINGS OF FITNESS THRESHOLD V

Test	RPSO			Optimum			Jin's			Fu's		
Problem	$V=40$	$V=45$	$V=50$	$V=40$	$V=45$	$V=50$	$V=40$	$V=45$	$V=50$	$V=40$	$V=45$	$V=50$
TP_{21}	2.09-	1.60-	1.12-	1.94-	1.48-	1.11-	3.19-	1.84-	0.91-	3.55	2.31	1.26
TP_{22}	1.10-	1.02	0.96+	1.11-	1.02	0.98+	0.78-	0.56-	0.45-	1.13	1.03	0.95
TP_{23}	1.29-	1.12-	0.98-	1.30-	1.09-	1.02+	0.97-	0.72-	0.50-	1.35	1.17	1.00
TP_{24}	1.27-	1.15-	1.08	1.38	1.21+	1.12+	1.12-	0.81-	0.60-	1.39	1.18	1.09
TP_{25}	1.34	0.95	0.91-	1.33	0.83-	0.83-	1.09-	0.85-	0.74-	1.34	0.96	0.92
TP_{26}	2.37-	1.74-	1.32-	2.25-	1.53-	1.19-	2.30-	1.77-	1.30-	2.52	1.85	1.38

3) Performance Measurement: In ROOT, our objective is to find solutions whose performances are robust against future environmental changes. At a particular time step, we are searching for solutions that are not only good for the current time step but also for future ones. Therefore, we can evaluate an algorithm's ROOT ability by evaluating the robustness (in this paper: average fitness and survival time) of solutions the algorithm found at each time step given a certain computational budget Δe at each time step. For the performance measurement, a solution's robustness is calculated using the true future fitness functions. Nonetheless, it should be noted that, in practice, we will not implement a new solution every time the environment changes due to the definition of average fitness and survival time. The performance measurement of ROOT used in this paper on a sequence of fitness functions $(f_1, f_2, \ldots f_N)$ is

$$\text{Performance}^{\text{ROOT}} = \frac{1}{N}\sum_{i=1}^{N} F(\mathbf{x}_i) \tag{24}$$

where $F(\mathbf{x}_i)$ is the robustness (either average fitness or survival time) of the solution $\mathbf{x}_i$ determined by an algorithm at time step i.

It should be noted that performance measurement for ROOT proposed here is dependent on parameter settings, being either T if average fitness is considered or V if survival time is investigated. Therefore, in order to compare an algorithm's ROOT ability comprehensively, results should be reported under different settings of T and V.

B. Experimental Results

Thirty independent runs are conducted, and all the results presented below are for time steps 20–100, i.e., from the 20th fitness function to the 100th fitness function in the sequence of TP_{ij}, $i = 1, 2$, $j = 1, 2, \ldots, 6$. The reason is that we require 20 length of time series data available for the prediction of a solution's future fitness in Jin's and Fu's. This means the performance measure in (24) is calculated by averaging the corresponding robustness of the solution determined by the investigated method at each time step (from time step 20–100). Since some solution's survival time defined in (4) can be infinity in certain benchmarks, we reset a solution's survival time to be 10 if its survival time is larger than 10.

Results with a "+" or "−" attached in the right hand side in Tables II and III are significantly better or worse than those of Fu's at a 0.05 significance level of Wilcoxon rank sum test.

1) Results for Maximizing Average Fitness: The results of maximizing average fitness in ROOT are presented in Table II. We can see that when the time window T takes a small value, i.e., $T = 2$, it is generally better to use TMO methods than methods specially designed for ROOT problems (i.e., Jin's and Fu's). The reason is straightforward as within a small time window T, previously good solutions tend to remain good during that time window. However, as the time window T gets larger ($T = 6, 10$), the advantage of ROOT methods (i.e., Jin's and Fu's, especially Fu's) over TMO methods gradually shows up. Fu's is significantly better than RPSO and optimum in almost all cases when $T = 6, 10$. However, for some specific dynamics in the DFFs, such as the random change in TP_{13} and the recurrent with noise change in TP_{16}, RPSO and optimum outperform Jin's and Fu's. The reason is that there is large randomness in the change of the DFFs in TP_{13} or TP_{16}, and hence it is difficult for prediction methods in Jin's and Fu's to predict a solution's future fitness well. As a result, the metric used in Jin's and Fu's may be a poor estimation of a solution's true robustness, which therefore degrades the ROOT performance of Jin's and Fu's. Finally, for the comparison between Jin's and Fu's, Fu's outperforms Jin's in most cases. The success of Fu's over Jin's is primarily due to the metric in Fu's, which guides EAs better to converge to good solutions. For more details of Fu's, readers are referred to [20].

2) Results for Maximizing Survival Time: The results for maximizing survival time in ROOT are presented in Table III. We can see that Fu's outperforms RPSO and optimum in most cases except in some cases when the fitness threshold

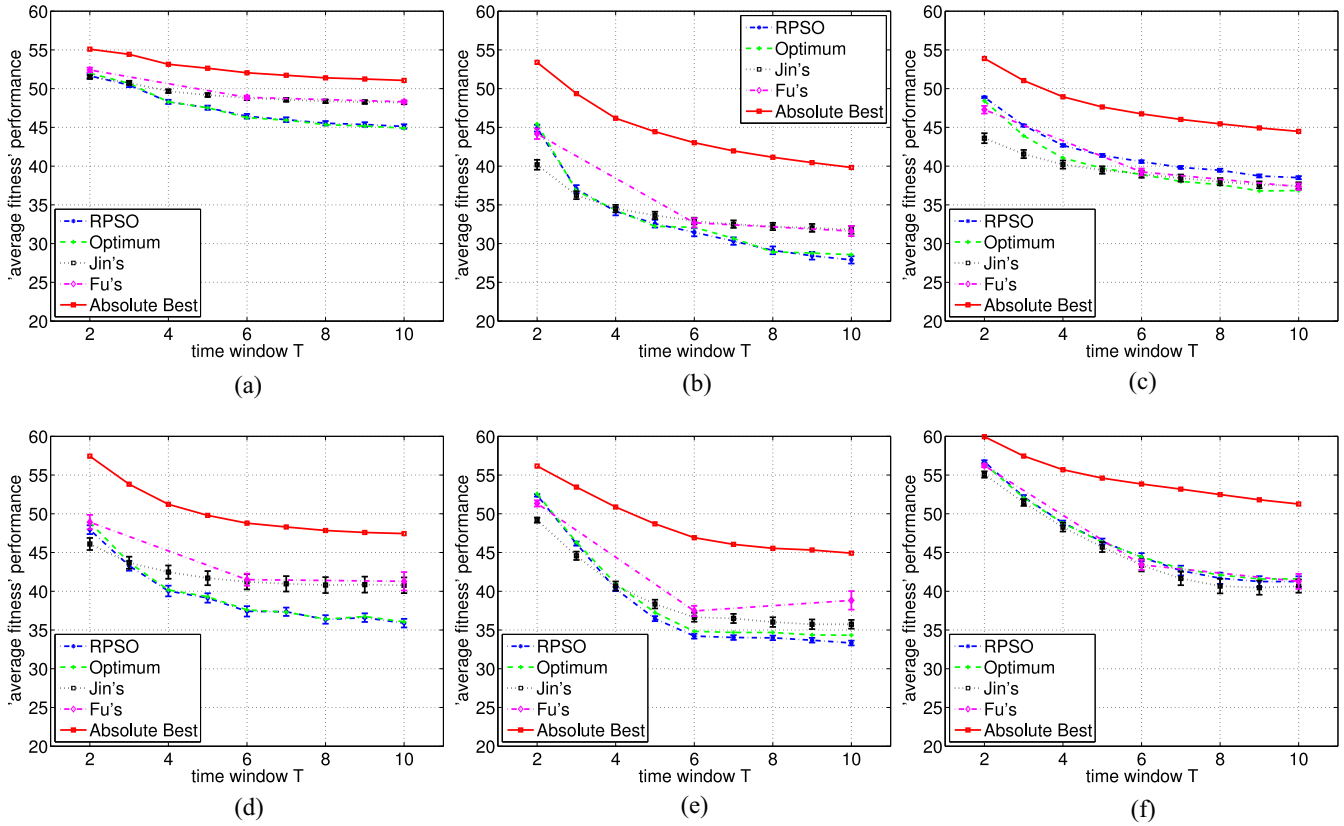


Fig. 5. Mean performance in (24) with one standard deviation errorbar over 30 runs of investigated methods under the robustness average fitness with different settings of time window T, in comparison with the corresponding absolute best performance. (a) TP_{11}. (b) TP_{12}. (c) TP_{13}. (d) TP_{14}. (e) TP_{15}. (f) TP_{16}.

V takes a large value ($V = 50$). The reason why TMO methods (i.e., RPSO and optimum) outperform Fu's in some cases when $V = 50$ is that the main difficulty in maximizing survival time is maximizing a solution's current fitness above V if V is set relatively high. In other words, maximizing survival time in ROOT is approximately equivalent to TMO when the fitness threshold V is set relatively high. The reason why Fu's outperforms RPSO and optimum in all other cases is that a solution's future fitness is considered in Fu's while TMO methods only maximize a solution's current fitness, without any consideration of a solution's future fitness. For the comparison between Jin's and Fu's, Fu's outperforms Jin's in most cases, and the primary reason is again due to the metric used in Fu's. For more details of Fu's, readers are referred to [20].

3) Gaps Between the Absolute Best and the Results Obtained by Investigated Methods: In Tables II and III, we have compared the ROOT performance of investigated methods on TP_{ij}, $i = 1, 2, 1 \leq j \leq 6$. We would also like to know the gaps between the absolute best performance and the performance of each investigated method for solving ROOT problems, based on which we can examine how well existing methods solve ROOT problems.

We plot the absolute best performance in terms of average fitness and the corresponding performance of investigated methods on TP_{1j} under different settings of time window T, $T = 2, 3, 4, \ldots, 10$, in Fig. 5. It should be noted that Fu's has only been tested on $T = 2, 6, 10$. Taking Fig. 5(a) for example, we calculate the absolute best performance in the following way (taking $T = 2$ for example). For TP_{11}, we have already generated a sequence of fitness functions of length 200: $(f_1^a, f_2^a, \ldots, f_{200}^a)$. We then calculate the absolute best average fitness in (6) starting from each fitness function in the sequence $(f_{20}^a, f_{21}^a, \ldots, f_{100}^a)$ based on Theorem 1. Finally, we average the absolute best average fitness starting from each fitness function in the sequence $(f_{20}^a, f_{21}^a, \ldots, f_{100}^a)$ to obtain the absolute best performance for TP_{11} with $T = 2$. Other absolute best performances for TP_{1j} $(1 \leq j \leq 6)$, with $T = 2, 3, \ldots, 10$, are produced in the same way. From Fig. 5, we can see that there is generally a large gap between the absolute best performance and the performance of each investigated method. Also, the gap gets larger as T increases. In other words, the difficulty in maximizing average fitness in ROOT increases with larger time window T in our experimental studies.

We plot the absolute best performance in terms of survival time and the corresponding performance of investigated methods on TP_{2j} under different settings of fitness threshold V, $V = 40, 41, 42, \ldots, 50$, in Fig. 6. It should be noted that Fu's has only been tested on $V = 40, 45, 50$. Taking Fig. 6(a) for example, we calculate the absolute best performance in the following way (taking $V = 40$ for example). For TP_{21}, we have already generated a sequence of fitness functions of length 200: $(f_1^s, f_2^s, \ldots, f_{200}^s)$. We then calculate the absolute best survival time in (7) starting from each fitness function in

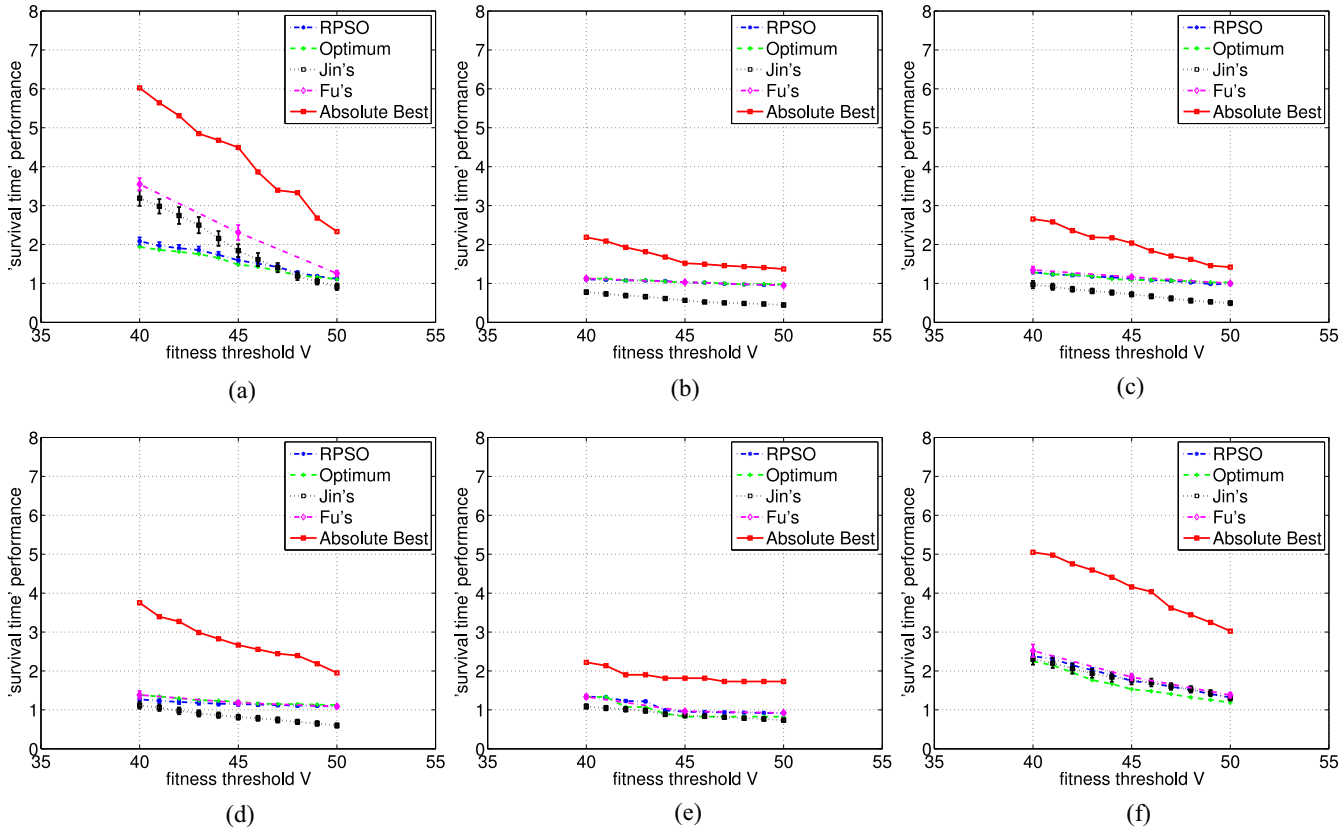


Fig. 6. Mean performance in (24) with one standard deviation errorbar over 30 runs of investigated methods under the robustness survival time with different settings of fitness threshold V, in comparison with the corresponding absolute best performance. (a) TP_{21}. (b) TP_{22}. (c) TP_{23}. (d) TP_{24}. (e) TP_{25}. (f) TP_{26}.

the sequence $(f^s_{20}, f^s_{21}, \ldots, f^s_{100})$ based on Theorem 2. Finally, we average the absolute best survival time starting from each fitness function in the sequence $(f^s_{20}, f^s_{21}, \ldots, f^s_{100})$ to obtain the absolute best performance for TP_{21} with $V = 40$. Other absolute best performances for TP_{2j} $(1 \leq j \leq 6)$, with $V = 40, 41, 42, \ldots, 50$, are produced in the same way. From Fig. 6, we can see that investigated methods do not perform well on ROOT problems when maximizing survival time as there is generally a large gap between the absolute best performance and the performance of each investigated method. Also, the gap gets larger as V decreases. In other words, the difficulty in maximizing survival time in ROOT increases as V gets smaller in our experimental studies.

VI. Conclusion

For the past decades, most research in dynamic optimization using EAs falls into the category of TMO. Recently, we proposed a new formulation, i.e., ROOT, which captures problem characteristics that cannot be captured by TMO. In this paper, we contribute to ROOT by developing two types of benchmarks that aim at testing an algorithm's ROOT abilities in terms of maximizing average fitness and survival time, respectively.

Our proposed benchmarks (i.e., RMPB-I and RMPB-II) are easily tunable and computationally efficient. Most importantly, both benchmarks allow the exact calculation of the absolute best average fitness and the absolute best survival time, respectively, which are extremely difficult to compute for most existing DOP benchmarks.

In this paper, we focus on discrete-time ROOT problems as most real-world DOPs, especially those that can be formulated as ROOT problems, are presented in a discrete-time manner. The assumptions behind ROOT and the task of ROOT methods are made explicit, which help to make ROOT as general and clear as possible and facilitate any further studies of ROOT. The task of a ROOT algorithm is to determine a solution that aims to maximize a corresponding robustness definition (i.e., average fitness or survival time in this paper) within a computational budget Δe after an environmental change occurs. Two major difficulties in solving ROOT problems are identified in this paper, which, to some extent, point out what a successful ROOT algorithm is required to be good at. Finally, several methods from the literature have been tested on RMPB-I and RMPB-II, which demonstrates potential strengths and weaknesses of different methods for ROOT problems with different dynamics. More importantly, the difficulties of ROOT problems are demonstrated by showing the gaps between the absolute best performances and those of investigated methods.

Research of ROOT is still in its infancy, and more attention should be given to ROOT for its practical advantage over TMO in the future. Firstly, in this paper we have discussed the single objective ROOT problem, i.e., either maximizing average fitness or survival time. It would be interesting to

define and explore multiple objectives for ROOT as many real-world DOPs are multiobjective in nature. Secondly, more successful ROOT methods are still needed, which can explicitly deal with the two major difficulties in ROOT, i.e., the difficulty to predict a solution's future fitness and the difficulty to evolve solutions based on inaccurate information. Only after being successfully tested on synthetic benchmarks, will ROOT methods be ready to be applied to real-world ROOT problems, and our RMPB benchmarks developed in this paper can server as proper synthetic benchmarks. Finally, investigating real-world DOPs from the perspective of ROOT would be very inspiring.

APPENDIX

Proof of Lemma 1

Remember that x_j and all the centers $c_{t+k}^{i_k j}$, $0 \le k \le T-1$, belong to the interval $[a, b]$. By definition, the MAF of a set of peak functions $\{\text{peak}_{t+k}^{i_k j} | 0 \le k \le T-1\}$ is $\max\{1/T \sum_{k=0}^{T-1} (h_{t+k}^{i_k j} - w_{t+k}^{i_k j} * |x_j - c_{t+k}^{i_k j}|) | x_j \in [a, b]\}$. Without loss of generality, suppose all the centers $c_{t+k}^{i_k j}$, $0 \le k \le T-1$, are in an ascending order in the list $(c_t^{i_0 j}, c_{t+1}^{i_1 j}, \ldots, c_{t+T-1}^{i_{T-1} j})$. It is easy to verify that $1/T \sum_{k=0}^{T-1} (h_{t+k}^{i_k j} - w_{t+k}^{i_k j} * |x_j - c_{t+k}^{i_k j}|)$ is monotonically increasing for the interval $[a, c_t^{i_0 j}]$ and monotonically decreasing for the interval $[c_{t+T-1}^{i_{T-1} j}, b]$. For any of the intervals $[c_{t+k-1}^{i_{k-1} j}, c_{t+k}^{i_k j}]$, $1 \le k \le T-1$, either $\sum_{l=0}^{k-1} w_{t+l}^{i_l j} < \sum_{l=k}^{T-1} w_{t+l}^{i_l j}$ is true or $\sum_{l=0}^{k-1} w_{t+l}^{i_l j} \ge \sum_{l=k}^{T-1} w_{t+l}^{i_l j}$ is true. This means $1/T \sum_{k=0}^{T-1} (h_{t+k}^{i_k j} - w_{t+k}^{i_k j} * |x_j - c_{t+k}^{i_k j}|)$ is monotonically either decreasing or increasing in the interval $[c_{t+k-1}^{i_{k-1} j}, c_{t+k}^{i_k j}]$. Therefore, we have that the MAF of a set of peak functions $\{\text{peak}_{t+k}^{i_k j} | 0 \le k \le T-1\}$ can be achieved when x_j takes the value of one of the centers $c_{t+k}^{i_k j}$, $0 \le k \le T-1$.

Proof of Lemma 2

By definition, the MAF of a set of dimensional functions $\{\text{dim}_{t+k}^{j} | 0 \le k \le T-1\}$ is $\max\{\frac{1}{T} \sum_{k=0}^{k=T-1} \max_{i=1}^{i=m} \{h_{t+k}^{ij} - w_{t+k}^{ij} * |x_j - c_{t+k}^{ij}|\} | x_j \in [a, b]\}$. Without loss of generality, suppose $\text{MAF}(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+T-1}^j)$ is achieved at point x_j^*. As a result, there exists a set of i_k^*s, $0 \le k \le T-1$, such that $\text{MAF}(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+T-1}^j)$ equals $\frac{1}{T} \sum_{k=0}^{k=T-1} (h_{t+k}^{i_k^* j} - w_{t+k}^{i_k^* j} * |x_j^* - c_{t+k}^{i_k^* j}|)$. $\frac{1}{T} \sum_{k=0}^{k=T-1} (h_{t+k}^{i_k^* j} - w_{t+k}^{i_k^* j} * |x_j^* - c_{t+k}^{i_k^* j}|)$ is no bigger than $\text{MAF}(\text{peak}_t^{i_0^* j}, \text{peak}_{t+1}^{i_1^* j}, \ldots, \text{peak}_{t+T-1}^{i_{T-1}^* j})$, which is no bigger tn $\max\{\text{MAF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+T-1}^{i_{T-1} j}) | 1 \le i_k \le m, 0 \le k \le T-1\}$.

On the other hand, without loss of generality, suppose $\max\{\text{MAF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+T-1}^{i_{T-1} j}) | 1 \le i_k \le m, 0 \le k \le T-1\}$ is obtained on a set of i'_ks, $0 \le k \le T-1$, such that $\max\{\text{MAF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+T-1}^{i_{T-1} j}) | 1 \le i_k \le m, 0 \le k \le T-1\}$ is equal to $\text{MAF}(\text{peak}_t^{i'_0 j}, \text{peak}_{t+1}^{i'_1 j}, \ldots, \text{peak}_{t+T-1}^{i'_{T-1} j})$. By definition, $\text{MAF}(\text{peak}_t^{i'_0 j}, \text{peak}_{t+1}^{i'_1 j}, \ldots, \text{peak}_{t+T-1}^{i'_{T-1} j})$ is $\max\{\frac{1}{T} \sum_{k=0}^{k=T-1} (h_{t+k}^{i'_k j} - w_{t+k}^{i'_k j} * |x_j - c_{t+k}^{i'_k j}|) | x_j \in [a, b]\}$, which is no bigger than $\max\{\frac{1}{T} \sum_{k=0}^{k=T-1} \max_{i=1}^{i=m} \{h_{t+k}^{ij} - w_{t+k}^{ij} * |x_j - c_{t+k}^{ij}|\} | x_j \in [a, b]\}$, i.e., $\text{MAF}(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+T-1}^j)$.

Therefore, we have Lemma 2 proved.

Proof of Theorem 1

By definition and arithmetic operation, we have

$$\begin{aligned}
&\text{MAF}\left(f_t^a, f_{t+1}^a, \ldots, f_{t+T-1}^a\right) \\
&\quad = \frac{1}{d} \sum_{j=1}^{d} \max \left\{ \frac{1}{T} \sum_{k=0}^{T-1} \max_{i=1}^{i=m} \left\{ h_{t+k}^{ij} \right.\right. \\
&\qquad \left.\left. - w_{t+k}^{ij} * \left| x_j - c_{t+k}^{ij} \right| \right\} \mid x_j \in [a, b] \right\} \\
&\quad = \frac{1}{d} \sum_{j=1}^{d} \text{MAF}\left(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+T-1}^j\right).
\end{aligned}$$

Based on Lemma 2, we have $\text{MAF}(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+T-1}^j)$ equal to $\max\{\text{MAF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+T-1}^{i_{T-1} j}) | 1 \le i_k \le m, 0 \le k \le T-1\}$. Therefore, we have Theorem 1 proved.

Proof of Lemma 3

Remember that x_j and all the centers $c_{t+k}^{i_k j}$, $0 \le k \le L-1$, belong to the interval $[a, b]$. Without loss of generality, suppose $\text{MIF}(\text{peak}_{t+p^*}^{i_{p^*} j}, \text{peak}_{t+q^*}^{i_{q^*} j})$ equals $\min\{\text{MIF}(\text{peak}_{t+p}^{i_p j}, \text{peak}_{t+q}^{i_q j}) | 0 \le p < q \le L-1\}$, and that $\text{MIF}(\text{peak}_{t+p^*}^{i_{p^*} j}, \text{peak}_{t+q^*}^{i_{q^*} j})$ is achieved at point $x_{p^* q^*}$. Furthermore, suppose that the center of $\text{peak}_{t+p^*}^{i_{p^*} j}$ is no bigger than the center of $\text{peak}_{t+q^*}^{i_{q^*} j}$: $c_{t+p^*}^{i_{p^*} j} \le c_{t+q^*}^{i_{q^*} j}$. It is easy to verify that $c_{t+p^*}^{i_{p^*} j} \le x_{p^* q^*} \le c_{t+q^*}^{i_{q^*} j}$.

For any peak function $\text{peak}_{t+l}^{i_l j}$ in the set $\{\text{peak}_{t+k}^{i_k j} | 0 \le k \le L-1\}$ whose center is no bigger than $x_{p^* q^*}$, suppose that $\text{MIF}(\text{peak}_{t+l}^{i_l j}, \text{peak}_{t+p^*}^{i_{p^*} j})$ is achieved at point $x_{l p^*}$, and that $\text{MIF}(\text{peak}_{t+l}^{i_l j}, \text{peak}_{t+q^*}^{i_{q^*} j})$ is achieved at point $x_{l q^*}$. On the one hand, if $x_{p^* q^*} = c_{t+p^*}^{i_{p^*} j}$, we have $\text{peak}_{t+p^*}^{i_{p^*} j}(c_{t+p^*}^{i_{p^*} j}) = \text{MIF}(\text{peak}_{t+p^*}^{i_{p^*} j}, \text{peak}_{t+q^*}^{i_{q^*} j}) \le \text{MIF}(\textit{peak}_{t+l}^{i_l j}, \text{peak}_{t+p^*}^{i_{p^*} j}) \le \text{peak}_{t+p^*}^{i_{p^*} j}(x_{l p^*})$, and hence $x_{p^* q^*} = x_{l p^*} = c_{t+p^*}^{i_{p^*} j}$. Therefore, $\text{peak}_{t+l}^{i_l j}(x_{p^* q^*}) = \text{peak}_{t+l}^{i_l j}(x_{l p^*}) \ge \text{MIF}(\text{peak}_{t+l}^{i_l j}, \text{peak}_{t+p^*}^{i_{p^*} j}) \ge \text{MIF}(\text{peak}_{t+p^*}^{i_{p^*} j}, \text{peak}_{t+q^*}^{i_{q^*} j})$. On the other hand, if $c_{t+p^*}^{i_{p^*} j} < x_{p^* q^*} \le c_{t+q^*}^{i_{q^*} j}$, we have $\text{peak}_{t+q^*}^{i_{q^*} j}(x_{p^* q^*}) = \text{MIF}(\text{peak}_{t+p^*}^{i_{p^*} j}, \text{peak}_{t+q^*}^{i_{q^*} j}) \le \text{MIF}(\text{peak}_{t+l}^{i_l j}, \text{peak}_{t+q^*}^{i_{q^*} j}) = \text{peak}_{t+q^*}^{i_{q^*} j}(x_{l q^*})$. Therefore, $x_{p^* q^*} \le x_{l q^*} \le c_{t+q^*}^{i_{q^*} j}$, and hence $\text{peak}_{t+l}^{i_l j}(x_{p^* q^*}) \ge \text{peak}_{t+l}^{i_l j}(x_{l q^*}) \ge \text{MIF}(\text{peak}_{t+l}^{i_l j}, \text{peak}_{t+q^*}^{i_{q^*} j}) \ge \text{MIF}(\text{peak}_{t+p^*}^{i_{p^*} j}, \text{peak}_{t+q^*}^{i_{q^*} j})$. To sum it all, we have proven, so far in this paragraph, that $\text{peak}_{t+l}^{i_l j}(x_{p^* q^*}) \ge \text{MIF}(\text{peak}_{t+p^*}^{i_{p^*} j}, \text{peak}_{t+q^*}^{i_{q^*} j})$ for any peak function $\text{peak}_{t+l}^{i_l j}$ in the set $\{\text{peak}_{t+k}^{i_k j} | 0 \le k \le L-1\}$ whose center is no bigger than $x_{p^* q^*}$.

Following the same proof procedure as above, it is easy to see that the following statement is true: for any peak function $\text{peak}_{t+r}^{i_r j}$ whose center, $c_{t+r}^{i_r j}$, is larger than $x_{p^*q^*}$, we have $\text{peak}_{t+r}^{i_r j}(x_{p^*q^*}) \geq \text{MIF}(\text{peak}_{t+p^*}^{i_{p^*} j}, \text{peak}_{t+q^*}^{i_{q^*} j})$. As a result, we have $\text{MIF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+L-1}^{i_{L-1} j}) \geq \text{MIF}(\text{peak}_{t+p^*}^{i_{p^*} j}, \text{peak}_{t+q^*}^{i_{q^*} j})$ proved.

On the other hand, $\text{peak}_{t+p^*}^{i_{p^*} j}$ and $\text{peak}_{t+q^*}^{i_{q^*} j}$ are a subset of all the peak functions $\{\text{peak}_{t+k}^{i_k j} | 0 \leq k \leq L-1\}$, and therefore $\text{MIF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+L-1}^{i_{L-1} j}) \leq \text{MIF}(\text{peak}_{t+p^*}^{i_{p^*} j}, \text{peak}_{t+q^*}^{i_{q^*} j})$.

Therefore, we have Lemma 3 proved.

Proof of Lemma 4

By definition, $\text{MIF}(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+L-1}^j)$ is $\max\{\min_{k=0}^{k=L-1} \max_{i=1}^{i=m}\{h_{t+k}^{ij} - w_{t+k}^{ij} * |x_j - c_{t+k}^{ij}|\} \mid x_j \in [a, b]\}$. Without loss of generality, suppose $\text{MIF}(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+L-1}^j)$ is achieved at point x_j^*. As a result, there exists a set of i_k^*s, $0 \leq k \leq L-1$, such that $\text{MIF}(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+L-1}^j)$ equals $\min_{k=0}^{k=L-1}(h_{t+k}^{i_k^* j} - w_{t+k}^{i_k^* j} * |x_j^* - c_{t+k}^{i_k^* j}|)$. $\min_{k=0}^{k=L-1}(h_{t+k}^{i_k^* j} - w_{t+k}^{i_k^* j} * |x_j^* - c_{t+k}^{i_k^* j}|)$ is no bigger than $\text{MIF}(\text{peak}_t^{i_0^* j}, \text{peak}_{t+1}^{i_1^* j}, \ldots, \text{peak}_{t+L-1}^{i_{L-1}^* j})$, which is no bigger than $\max\{\text{MIF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+L-1}^{i_{L-1} j}) \mid 1 \leq i_k \leq m, 0 \leq k \leq L-1\}$.

On the other hand, without loss of generality, suppose $\max\{\text{MIF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+L-1}^{i_{L-1} j}) \mid 1 \leq i_k \leq m, 0 \leq k \leq L-1\}$ is obtained on a set of i'_ks, $0 \leq k \leq L-1$, such that $\max\{\text{MIF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+L-1}^{i_{L-1} j}) \mid 1 \leq i_k \leq m, 0 \leq k \leq L-1\}$ equals $\text{MIF}(\text{peak}_t^{i'_0 j}, \text{peak}_{t+1}^{i'_1 j}, \ldots, \text{peak}_{t+L-1}^{i'_{L-1} j})$. By definition, $\text{MIF}(\text{peak}_t^{i'_0 j}, \text{peak}_{t+1}^{i'_1 j}, \ldots, \text{peak}_{t+L-1}^{i'_{L-1} j})$ is $\max\{\min_{k=0}^{k=L-1}(h_{t+k}^{i'_k j} - w_{t+k}^{i'_k j} * |x_j - c_{t+k}^{i'_k j}|) \mid x_j \in [a, b]\}$, which is no bigger than $\max\{\min_{k=0}^{k=L-1} \max_{i=1}^{i=m}\{h_{t+k}^{ij} - w_{t+k}^{ij} * |x_j - c_{t+k}^{ij}|\} \mid x_j \in [a, b]\}$, i.e., $\text{MIF}(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+L-1}^j)$.

Therefore, we have Lemma 4 proved.

Proof of Theorem 2

For any dimension j, $1 \leq j \leq d$, $\text{MIF}(f_t^s, f_{t+1}^s, \ldots, f_{t+L-1}^s) \leq \text{MIF}(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+L-1}^j)$, since $f_{t+k}^s \leq \text{dim}_{t+k}^j$, $0 \leq k \leq T-1$. Supposing $\text{MIF}(\text{dim}_t^{j^*}, \text{dim}_{t+1}^{j^*}, \ldots, \text{dim}_{t+L-1}^{j^*}) = \min_{j=1}^d \text{MIF}(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+L-1}^j)$, we have $\text{MIF}(f_t^s, f_{t+1}^s, \ldots, f_{t+L-1}^s) \leq \text{MIF}(\text{dim}_t^{j^*}, \text{dim}_{t+1}^{j^*}, \ldots, \text{dim}_{t+L-1}^{j^*})$.

On the other hand, suppose for any dimension j, $1 \leq j \leq d$, $\text{MIF}(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+L-1}^j)$ is achieved at point x_j^*: $\text{MIF}(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+L-1}^j) = \min\{\text{dim}_{t+i}^j(x_j^*) | 0 \leq i \leq L-1\}$. By denoting $\mathbf{x}^* = (x_1^*, x_2^*, \ldots, x_d^*)$, we have $\min\{f_{t+i}^s(\mathbf{x}^*) | 0 \leq i \leq L-1\} = \text{MIF}(\text{dim}_t^{j^*}, \text{dim}_{t+1}^{j^*}, \ldots, \text{dim}_{t+L-1}^{j^*})$. This means $\text{MIF}(f_t^s, f_{t+1}^s, \ldots, f_{t+L-1}^s) \geq \min\{f_{t+i}^s(\mathbf{x}^*) | 0 \leq i \leq L-1\} = \text{MIF}(\text{dim}_t^{j^*}, \text{dim}_{t+1}^{j^*}, \ldots, \text{dim}_{t+L-1}^{j^*})$.

Therefore, we have $\text{MIF}(f_t^s, f_{t+1}^s, \ldots, f_{t+L-1}^s)$ equal to $\text{MIF}(\text{dim}_t^{j^*}, \text{dim}_{t+1}^{j^*}, \ldots, \text{dim}_{t+L-1}^{j^*})$.

Based on Lemma 4, we have $\text{MIF}(\text{dim}_t^j, \text{dim}_{t+1}^j, \ldots, \text{dim}_{t+L-1}^j)$ equal to $\max\{\text{MIF}(\text{peak}_t^{i_0 j}, \text{peak}_{t+1}^{i_1 j}, \ldots, \text{peak}_{t+L-1}^{i_{L-1} j}) \mid 1 \leq i_k \leq m, 0 \leq k \leq L-1\}$. Therefore, we have Theorem 2 proved.

REFERENCES

[1] J. Branke, *Evolutionary Optimization in Dynamic Environments*, vol. 3. Boston, MA, USA: Kluwer, 2002.

[2] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments—A survey," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 303–317, Jun. 2005.

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

[4] T. Blackwell, J. Branke, and X. Li, "Particle swarms for dynamic optimization problems," in *Swarm Intelligence*. Berlin, Germany: Springer, 2008, pp. 193–217.

[5] T. Nguyen and X. Yao, "Benchmarking and solving dynamic constrained problems," in *Proc. 2009 IEEE Congr. Evol. Comput. (CEC)*, Trondheim, Norway, pp. 690–697.

[6] X. Yu, K. Tang, T. Chen, and X. Yao, "Empirical analysis of evolutionary algorithms with immigrants schemes for dynamic optimization," *Memetic Comput.*, vol. 1, no. 1, pp. 3–24, 2009.

[7] S. Yang and C. Li, "A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 14, no. 6, pp. 959–974, Dec. 2010.

[8] P. Rohlfshagen and X. Yao, "Dynamic combinatorial optimisation problems: An analysis of the subset sum problem," *Soft Comput.*, vol. 15, no. 9, pp. 1723–1734, 2011.

[9] P. Stroud, "Kalman-extended genetic algorithm for search in nonstationary environments with noisy fitness evaluations," *IEEE Trans. Evol. Comput.*, vol. 5, no. 1, pp. 66–77, Feb. 2001.

[10] R. Ursem, T. Krink, M. Jensen, and Z. Michalewicz, "Analysis and modeling of control tasks in dynamic systems," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 378–389, Aug. 2002.

[11] C. Rossi, M. Abderrahim, and J. Díaz, "Tracking moving optima using Kalman-based predictions," *Evol. Comput.*, vol. 16, no. 1, pp. 1–30, 2008.

[12] H. Handa, L. Chapman, and X. Yao, "Dynamic salting route optimisation using evolutionary computation," in *Proc. 2005 IEEE Congr. Evol. Comput.*, vol. 1. Edinburgh, U.K., pp. 158–165.

[13] L. T. Bui, Z. Michalewicz, E. Parkinson, and M. B. Abello, "Adaptation in dynamic environments: A case study in mission planning," *IEEE Trans. Evol. Comput.*, vol. 16, no. 2, pp. 190–209, Apr. 2012.

[14] A. Larsen and O. B. Madsen, "The dynamic vehicle routing problem," Ph.D. dissertation, Dept. Transport, Inst. Transport Logist. (ITS), Tech. Univ. Denmark, Lyngby, Denmark, 2000.

[15] H. Handa, "Fitness function for finding out robust solutions on time-varying functions," in *Proc. 8th ACM Annu. Conf. Genet. Evol. Comput.*, Seattle, WA, USA, 2006, pp. 1195–1200.

[16] J. A. 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," *J. Sched.*, vol. 11, no. 5, pp. 323–346, 2008.

[17] D. E. Wilkins, S. F. Smith, L. A. Kramer, T. J. Lee, and T. W. Rauenbusch, "Airlift mission monitoring and dynamic rescheduling," *Engi. Appl. Artif. Intell.*, vol. 21, no. 2, pp. 141–155, 2008.

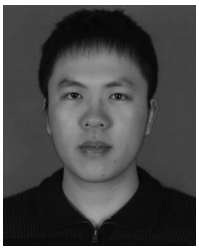
[18] X. Yu, Y. Jin, K. Tang, and X. Yao, "Robust optimization over time—A new perspective on dynamic optimization problems," in *Proc. 2010 IEEE Congr. Evol. Comput.*, Barcelona, Spain, pp. 1–6.

[19] H. Fu, B. Sendhoff, K. Tang, and X. Yao, "Characterizing environmental changes in robust optimization over time," in *Proc. 2012 IEEE Congr. Evol. Comput.*, Brisbane, QLD, Australia, pp. 1–8.

[20] H. Fu, B. Sendhoff, K. Tang, and X. Yao, "Finding robust solutions to dynamic optimization problems," in *Proc. 16th Eur. Conf. Appl. Evol. Comput.*, Vienna, Austria, 2013, pp. 616–625.

[21] H. G. Cobb, "An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments," Naval. Res. Lab., Navy Center Appl. Res. Artif. Intell., Washington, DC, USA, Tech. Rep. 6760, 1990.

[22] J. Lewis, E. Hart, and G. Ritchie, "A comparison of dominance mechanisms and simple mutation on non-stationary problems," in *Parallel Problem Solving from Nature—PPSN V*. Berlin, Germany: Springer, 1998, pp. 139–148.
[23] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *Proc. 1999 IEEE Congr. Evol. Comput.*, vol. 3. Washington, DC, USA, pp. 1875–1882.
[24] R. W. Morrison and K. A. De Jong, "A test problem generator for non-stationary environments," in *Proc. 1999 IEEE Congr. Evol. Comput.*, vol. 3. Washington, DC, USA, pp. 1859–1866.
[25] Y. Jin and B. Sendhoff, "Constructing dynamic optimization test problems using the multi-objective optimization concept," in *Applications of Evolutionary Computing*. Berlin, Germany: Springer, 2004, pp. 525–536.
[26] C. Li *et al.*, "Benchmark generator for CEC 2009 competition on dynamic optimization," Dept. Comput. Sci., Univ. Leicester, Leicester, U.K., Univ. Birmingham, Birmingham, U.K., Nanyang Technol. Univ., Singapore, Tech. Rep., 2008.
[27] S. Yang, "Non-stationary problem optimization using the primal-dual genetic algorithm," in *Proc. 2003 IEEE Congr. Evol. Comput.*, vol. 3. Canberra, ACT, Australia, pp. 2246–2253.
[28] R. Tinós and S. Yang, "Continuous dynamic problem generators for evolutionary algorithms," in *Proc. 2007 IEEE Congr. Evol. Comput.*, Singapore, pp. 236–243.
[29] T. T. Nguyen, "Continuous dynamic optimisation using evolutionary algorithms," Ph.D. dissertation, School Comput. Sci., Univ. Birmingham, Birmingham, U.K., 2011.
[30] P. A. N. Bosman, "Learning, anticipation and time-deception in evolutionary online dynamic optimization," in *Proc. 2005 ACM Workshops Genet. Evol. Comput.*, Washington, DC, USA, pp. 39–47.
[31] H. Akaike, "Fitting autoregressive models for prediction," *Ann. Inst. Stat. Math.*, vol. 21, no. 1, pp. 243–247, 1969.
[32] G. Box and G. Jenkins, *Time Series Analysis: Forecasting and Control*. Englewood Cliffs, NJ, USA: Prentice Hall, 1994.
[33] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Comput.*, vol. 9, no. 1, pp. 3–12, 2005.
[34] S. Ratschan, "Efficient solving of quantified inequality constraints over the real numbers," *ACM Trans. Comput. Logic (TOCL)*, vol. 7, no. 4, pp. 723–748, 2006.
[35] P. Nuzzo, A. Puggelli, S. A. Seshia, and A. Sangiovanni-Vincentelli, "CalCS: SMT solving for non-linear convex constraints," in *Proc. 2010 IEEE Form. Methods Comput.-Aided Design (FMCAD)*, Lugano, Switzerland, pp. 71–80.
[36] Y. Jin, K. Tang, X. Yu, B. Sendhoff, and X. Yao, "A framework for finding robust optimal solutions over time," *Memetic Comput.*, vol. 5, no. 1, pp. 3–18, 2013.
[37] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 1994.
[38] M. Clerc and J. Kennedy, "The particle swarm—Explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 58–73, Feb. 2002.



Haobo Fu received the Ph.D. degree in computer science from University of Birmingham, Birmingham, U.K., in 2014.

Since 2014 he has been a Research and Development Software Engineer with the Big Data Laboratory, Baidu, Inc., Beijing, China. His research interests include evolutionary dynamic optimization and machine learning. He has published several papers on evolutionary dynamic optimization, especially about finding robust solutions in dynamic environments.

Dr. Fu received the Full Ph.D. Scholarship from Honda Research Institute Europe and the 2012 IEEE Computational Intelligence Society Students Travel Grants.



Bernhard Sendhoff (SM'05) received the Ph.D. degree in applied physics from Ruhr-Universität Bochum, Bochum, Germany, in 1998.

From 1999 to 2002 he was with Honda Research and Development Europe GmbH, Offenbach, Germany, and since 2003 he has been with the Honda Research Institute Europe GmbH. Since 2007 and 2008 he has been an Honorary Professor with the School of Computer Science, University of Birmingham, Birmingham, U.K., and the Technical University of Darmstadt, Darmstadt, Germany, respectively. Since 2011 he has been the President of Honda Research Institute Europe GmbH. His research interests include methods from computational intelligence and their applications in development, production, and services. He has authored or co-authored over 150 scientific papers and over 30 patents.

Prof. Sendhoff is a Senior Member of ACM.



Ke Tang (M'07–SM'13) received the B.Eng. degree from Huazhong University of Science and Technology, Wuhan, China, and the Ph.D. degree from Nanyang Technological University, Singapore, in 2002 and 2007, respectively.

Since 2007 he has been with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China, where he is currently a Professor. His research interests include computational intelligence, evolutionary computation, machine learning, and their real-world applications. He has authored or co-authored over 90 refereed publications.

Dr. Tang is an Associate Editor of IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE, *Computational Optimization and Applications Journal*, and *Frontiers of Computer Science Journal*.



Xin Yao (F'03) received the B.Sc. in 1982 from the University of Science and Technology of China (USTC), M.Sc. in 1985 from North China Institute of Computing Technologies, and Ph.D. degree in 1990 from USTC.

He is a Chair (Professor) of Computer Science and the Director of the Centre of Excellence for Research in Computational Intelligence and Applications, University of Birmingham, Birmingham, U.K. His research interests include evolutionary computation, ensemble learning, and their applications. He has over 450 refereed publications in international journals and conferences.

He received the 2001 IEEE Donald G. Fink Prize Paper Award, the 2010 IEEE Transactions on Evolutionary Computation Outstanding Paper Award, the 2010 BT Gordon Radley Award for Best Author of Innovation (Finalist), the 2011 IEEE Transactions on Neural Networks Outstanding Paper Award, the Prestigious Royal Society Wolfson Research Merit Award in 2012, the IEEE Computational Intelligence Society (CIS) Evolutionary Computation Pioneer Award in 2013, and several other best paper awards. He was the former Editor-in-Chief of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION from 2003 to 2008. He will be the President of the IEEE Computational Intelligence Society, from 2014 to 2015.