Robust Optimization Over Time: A Critical Review

Danial Yazdani, *Member, IEEE,* Mohammad Nabi Omidvar, *Senior member, IEEE,* Donya Yazdani, Jürgen Branke, Trung Thanh Nguyen, Amir H. Gandomi, *Senior member, IEEE,* Yaochu Jin, *Fellow, IEEE,* and Xin Yao, *Fellow, IEEE*

***Abstract*—Robust optimization over time (ROOT) is the combination of robust optimization and dynamic optimization. In ROOT, frequent changes to deployed solutions are undesirable, which can be due to the high cost of switching between deployed solutions, limitations on the resources required to deploy new solutions, and/or the system's inability to tolerate frequent changes in the deployed solutions. ROOT is dedicated to the study and development of algorithms capable of dealing with the implications of deploying or maintaining solutions over longer time horizons involving multiple environmental changes. This paper presents an in-depth review of the research on ROOT. The overarching aim of this survey is to help researchers gain a broad perspective on the current state of the field, what has been achieved so far, and the existing challenges and pitfalls. This survey also aims to improve accessibility and clarity by standardizing terminology and unifying mathematical notions used across the field, providing explicit mathematical formulations of definitions, and improving many existing mathematical descriptions. Moreover, we classify ROOT problems based on two ROOT-specific criteria: the requirements for changing or keeping deployed solutions and the number of deployed solutions. This classification helps researchers gain a better understanding of the characteristics and requirements of ROOT problems, which is crucial to systematic algorithm design and benchmarking. Additionally, we classify ROOT methods based on the approach they use for finding robust solutions and provide a comprehensive review of them. This survey also reviews ROOT benchmarks and performance indicators. Finally, we identify several future research directions.**

***Index Terms*—Robust optimization over time, dynamic optimization problems, robust optimization, optimization, evolutionary algorithms.**

Danial Yazdani is with the Faculty of Engineering & Information Technology, University of Technology Sydney, Ultimo 2007, Australia. (e-mail: danial.yazdani@gmail.com)

Mohammad Nabi Omidvar is with the School of Computing, University of Leeds, and Leeds University Business School, Leeds LS2 9JT, United Kingdom. (e-mail: m.n.omidvar@leeds.ac.uk)

Donya Yazdani is with the Department of Computer Science, University of Sheffield, Sheffield S1 4DP, United Kingdom. (e-mail: dyazdani1@sheffield.ac.uk)

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

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

Amir H. Gandomi is with the Faculty of Engineering & Information Technology, University of Technology Sydney, Ultimo 2007, Australia. He is also with the University Research and Innovation Center (EKIK), Obuda University, Budapest 1034, Hungary. (e-mail: Gandomi@uts.edu.au)

Yaochu Jin is with the Faculty of Technology, Bielefeld University, Bielefeld 33615, Germany. (e-mail: yaochu.jin@uni-bielefeld.de)

Xin Yao is with the Research Institute of Trustworthy Autonomous Systems (RITAS), and Guangdong Provincial Key Laboratory of Brain inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China. He is also with the CERCIA, School of Computer Science, Birmingham B15 2TT, United Kingdom. (e-mail: xiny@sustech.edu.cn)

I. Introduction

FINDING robust solutions is an important aspect of many optimization problems with many practical ramifications [1]. In many situations, the theoretical global optimum may not necessarily coincide with the desired solution due to modeling approximations and environmental uncertainties. Temporal changes of the environmental parameters can also change the previous conditions under which a solution is obtained. Other considerations such as maintenance and sustainability implications and the overall operational cost may also make a sub-optimal solution preferable over the global optimum.

The goal of robust optimization is to guard against sensitivity to fluctuations caused by modeling restrictions and environmental uncertainties which are pervasive in a wide range of application areas. In linear programming for instance, the notion of sensitivity or post-optimal analysis is primarily concerned with finding robust solutions to guard against drastic changes in the solution quality caused by changes in the coefficients of the objective function or its constraints [2]. In machine learning, it has been shown that robustness is tightly linked to generalizability [3]. In multidisciplinary design optimization [4], the subsystems of a larger complex system are optimized independently due to prohibitive complexity of the system as a whole and the existence of potential time constraints. The inevitable coupling of the subsystems makes robust optimization the cornerstone of a seamless integration [5].

In a certain type of optimization known as dynamic optimization, uncertainties result from environmental changes. A dynamic optimization problem (DOP) can be defined as:

$$\left\{f\left(\mathbf{x},\alpha^{(t)}\right)\right\}_{t=1}^{t_{\max}}=\left\{f\left(\mathbf{x},\alpha^{(1)}\right),f\left(\mathbf{x},\alpha^{(2)}\right),\ldots,f\left(\mathbf{x},\alpha^{(t_{\max})}\right)\right\}, \tag{1}$$

where f is the dynamic objective function[1], $t\in[1,t_{\max}]$ is the time index, $\mathbf{x}$ is a solution in the search space, and α is a set of time-dependent control parameters[2] of the objective function. The uncertainties caused by environmental changes in (1) influence the system via the α-variables [6] which cannot be removed/relaxed and must be considered in the optimization process [7]. Note that in the DOP literature, it is assumed that there is a degree of similarity between successive environments, which is prevalent in real-world applications [8, 9]. If the degree of similarity between successive environments is too low, nothing can be learned from the past; therefore,

[1]For the sake of brevity, in the rest of this survey, we use $f^{(t)}(\mathbf{x})$ instead of $f\left(\mathbf{x},\alpha^{(t)}\right)$ to represent the objective function in the tth environment.

[2]Also known as "environmental parameters."

using algorithms that work based on historical information is ineffective. The only way to tackle such problems is to re-optimize by restarting the optimization algorithm after each environmental change [10].

Indeed, in DOPs, the magnitude of temporal changes to the objective function is so large that the conventional robust optimization methods cannot be used to handle the uncertainties caused by environmental changes. This creates a whole host of new problems pertaining to the notion of robustness unique to dynamic optimization. In these problems, finding and tracking of global optima across environmental changes is of concern, and the implications and limitations of deploying new solutions plays a central role in maintaining robustness across all environmental changes. In these problems, robust solutions do not remain desirable forever and they need to be changed after some environmental changes. Robust optimization over time (ROOT) [11, 12] integrates the principles of both robust optimization and dynamic optimization and is dedicated to the study and development of algorithms capable of dealing with the implications of deploying or maintaining robust solutions over longer time horizons involving multiple environmental changes.

In this paper, we provide an in-depth and critical review of the research in ROOT since it was first introduced in 2010 [11]. Our investigation has shown that the inherent ambiguity in the definition of robustness has caused some confusion in the field, resulting in biased comparisons due to the incorrect use of performance indicators and incongruent comparisons of ROOT algorithms. We attempt to rectify this by putting forward two taxonomies of ROOT problems and algorithms, a detailed investigation of benchmarks and performance indicators, providing mathematical expressions for many definitions that have not been previously formulated, improving many existing mathematical descriptions, standardizing the terminology, unifying the mathematical notations, clarifying several misconceptions, and highlighting the pitfalls and challenges facing the field.

The organization of the remainder of this article is as follows. Section II reviews the ROOT problems. ROOT methods are reviewed in Section III. We discuss the benchmark problems and performance indicators used in the ROOT literature in Section IV. Section V provides several potential future research directions. Finally, this survey is concluded in Section VI.

II. ROOT PROBLEMS

In this survey, we classify ROOT problems based on two different aspects:

1) Requirements for changing or keeping deployed solutions:
   - ROOT problems with a *quality threshold* for determining the acceptability of the deployed solution(s) ($\mathrm{ROOT_Q}$),
   - ROOT problems with a time-window-based *temporal threshold* for deploying a new solution(s) ($\mathrm{ROOT_T}$), and
   - ROOT problems in which the benefit of switching the deployed solution(s) is jointly considered with its/their acceptability ($\mathrm{ROOT_G}$).
2) Number of deployed solution(s):
   - ROOT problems in which there is one and only one deployed solution at each point in time ($\mathrm{ROOT^S}$), and
   - ROOT problems in which there are multiple deployed solutions ($\mathrm{ROOT^M}$).

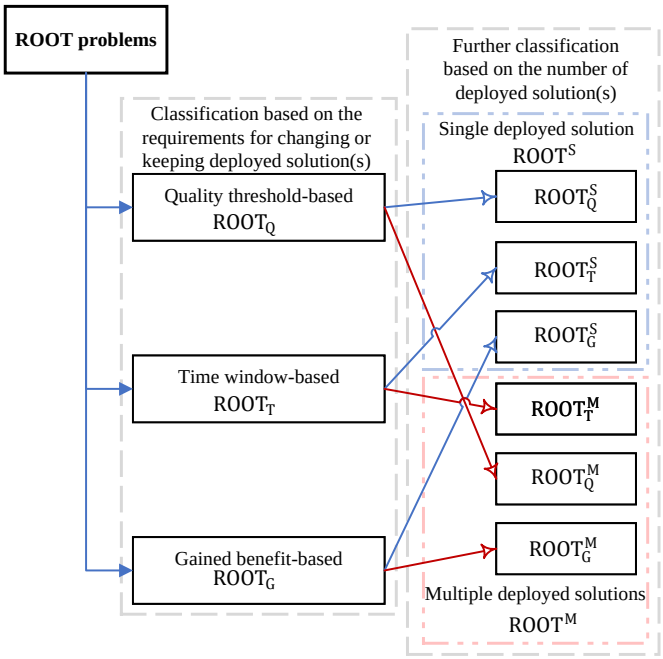


Fig. 1. Taxonomy of the ROOT problems based on the solution deployment considerations.

In the proposed taxonomy[3], we have combined these two major aspects, resulting in classifying ROOT problems into six different classes, which are shown in Fig. 1. Among different classes of ROOT problems, $\mathrm{ROOT_T^M}$ and $\mathrm{ROOT_G^M}$ have not been investigated so far, which will be discussed in Section V-A as future work. Below, we describe $\mathrm{ROOT_Q^S}$ [6, 11], $\mathrm{ROOT_T^S}$ [13], $\mathrm{ROOT_G^S}$ [14], and $\mathrm{ROOT_Q^M}$ [15] problems, which have been investigated in the literature.

A. $\mathbf{ROOT_Q^S}$

$\mathrm{ROOT_Q^S}$, which is a class of ROOT problems with a single deployed solution at each point in time and a quality threshold for determining the acceptability of the deployed solution, has been defined by Yu et al. [11] as the first class of ROOT problems investigated in the literature. In $\mathrm{ROOT_Q^S}$ problems, the main goal is to maximize the average number of successive environments where a deployed solution's fitness value remains acceptable [11, 12]. In these problems, a deployed

[3]The notation, which we are using in the proposed taxonomy for showing different classes of ROOT problems, uses the superscript to refer to the number of deployed solutions, where 'S' and 'M' refer to single and multiple deployed solutions, respectively. Besides, the subscript in the notation is reserved for the requirements for changing or keeping the deployed solutions, which are quality-threshold-based ('Q'), temporal-threshold-based ('T'), and gained-benefit-based ('G').

solution will be reused in successive environments until its quality drops below an acceptable level in a new environment. In real-world applications, $\text{ROOT}^{\text{S}}_{\text{Q}}$ problems are the ones in which "frequent changes" of the deployed solutions violate the system requirements and result in instability of the system or deterioration of some important factors, such as safety, customer satisfaction, and costs. In such circumstances, it is preferable to keep the deployed solution as long as it remains acceptable. Note that the acceptable frequency of changes in the deployed solutions can vary depending on the problem and the specific requirements of the application, such as the time required to adapt to a newly deployed solution, the stability of the system, the cost of deploying a new solution, denoted as the switching cost, and the impact of changes in the deployed solution on other parts of the system.

Given a DOP $f^{(t)}(\mathbf{x})$ with $t_{\max}$ environments, the aim of $\text{ROOT}^{\text{S}}_{\text{Q}}$ is to find a set of deployed solutions $\mathcal{S} = \{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_{|\mathcal{S}|}\}$ where $|\mathcal{S}|$ is the total number of the deployed solutions over $t_{\max}$ environments and $1 \leq |\mathcal{S}| \leq t_{\max}$. Assume $\mathbf{s}_i \in \mathcal{S}$ is deployed in the b_ith environment, $\mathfrak{a}^{(t)}(\cdot)$ is a function with $\{0, 1\}$ output that defines the acceptability of a solution based on the quality threshold in the tth environment[4], where $\mathfrak{a}^{(t)}(\mathbf{s}_i) = 1$ indicates that $\mathbf{s}_i$ is acceptable in the tth environment, and 0 otherwise, and n_i is the number of environmental changes beyond which $\mathbf{s}_i$ remains acceptable. Therefore, a deployed solution $\mathbf{s}_i$ remains operational for n_i environments if $\mathfrak{a}^{(b_i)}(\mathbf{s}_i) = 1, \mathfrak{a}^{(b_i+1)}(\mathbf{s}_i) = 1, \cdots, \mathfrak{a}^{(b_i+n_i)}(\mathbf{s}_i) = 1$. If $\mathfrak{a}^{(b_i+n_i+1)}(\mathbf{s}_i) = 0$, then the next solution $\mathbf{s}_{i+1}$ is deployed in b_{i+1}th environment, where $b_{i+1} = b_i + n_i + 1$. A deployed solution $\mathbf{s}_i$ is called a robust solution if its quality remains acceptable after at least one environmental change, i.e., $n_i > 0$ or at least $\mathfrak{a}^{(b_i+1)}(\mathbf{s}_i) = 1$.

B. $\mathbf{ROOT^S_T}$

Another class of ROOT problems, $\text{ROOT}^{\text{S}}_{\text{T}}$, has been introduced by Fu et al. [13], representing real-world problems in which the system cannot handle/tolerate changes in the deployed solution sooner than a predefined time threshold. Consequently, a time window is defined in this class of ROOT problems during which the deployed solution must remain deployed (i.e., operational). In other words, in $\text{ROOT}^{\text{S}}_{\text{T}}$, a solution must be chosen for deployment only at the beginning of each time window.

$\text{ROOT}^{\text{S}}_{\text{T}}$ has a temporal threshold called time window. In the ROOT literature, each time window starts at the beginning of an environment and lasts for $t_{\text{win}} \in \{2, 3, \cdots\}$ environments. In $\text{ROOT}^{\text{S}}_{\text{T}}$ problems, a new solution must be chosen for deployment at the first environment of each time window. Each deployed solution will be kept until the end of the time window. Given a DOP $f^{(t)}(\mathbf{x})$ with $t_{\max}$ environments and a set of time windows $\mathcal{T} = \{t_{\text{win},1}, t_{\text{win},2}, \cdots, t_{\text{win},|\mathcal{T}|}\}$, the goal of the problem is to find a set of solutions $\mathcal{S} = \{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_{|\mathcal{T}|}\}$ that maximizes the average fitness over all environments. Herein, $\mathbf{s}_i$ is the deployed solution in the ith time window, which begins at the beginning of the b_ith environment and ends at the end of the $(b_i + t_{\text{win},i} - 1)$th environment. Thus, in $\text{ROOT}^{\text{S}}_{\text{T}}$, the main goal is to choose solutions for deployment during the first environment of each time window such that:

$$\text{Maximize}: \ \frac{1}{t_{\max}} \sum_{i=1}^{|\mathcal{T}|} \sum_{j=0}^{(t_{\text{win},i}-1)} f^{(y_i+j)}(\mathbf{s}_i), \tag{2}$$

where

$$y_i = \begin{cases} 1, & \text{if } i = 1 \\ \sum_{k=1}^{i-1} t_{\text{win},k} & \text{otherwise} \end{cases}. \tag{3}$$

Note that unlike $\text{ROOT}^{\text{S}}_{\text{Q}}$ in which there is a quality threshold to define acceptability of the deployed solutions, there is no such a threshold in $\text{ROOT}^{\text{S}}_{\text{T}}$ and the goal is to maximize the average of the fitness values over each time window. It is worth mentioning that in the ROOT literature, only a special case of $\text{ROOT}^{\text{S}}_{\text{T}}$ problems has been investigated in which the length of all time windows are equal and fixed over time [13].

C. $\mathbf{ROOT^S_G}$

Yazdani et al. [14] identified another class of ROOT problems, called $\text{ROOT}^{\text{S}}_{\text{G}}$, where the gained benefit of switching the deployed solution is taken into account. In $\text{ROOT}^{\text{S}}_{\text{G}}$, a quality threshold is also considered for determining acceptability of the deployed solutions. Therefore, a new solution *must* be chosen to replace the current deployed solution when it becomes unacceptable. Besides, the deployed solution, which is still acceptable, *can* be replaced by a new solution whose fitness is significantly higher than that of the deployed solution, making the benefit of switching outweighs the cost. If such a qualified solution is found, regardless of the acceptability of the current deployed solution, this new solution will be deployed.

The goal of $\text{ROOT}^{\text{S}}_{\text{G}}$ is to find a set of deployed solutions $\mathcal{S} = \{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_{|\mathcal{S}|}\}$ that maximizes the total benefit, which is defined in [14] as the total fitness minus total switching costs across all environments. Assuming that the deployed solution $\mathbf{s}_i \in \mathcal{S}$ is used from b_ith to $(b_i + n_i)$th environment, the goal of $\text{ROOT}^{\text{S}}_{\text{G}}$ can be defined as:

$$\text{Maximize}: \ \sum_{i=1}^{|\mathcal{S}|} \sum_{j=b_i}^{(b_i+n_i)} f^{(j)}(\mathbf{s}_i) - \sum_{i=1}^{(|\mathcal{S}|-1)} \mathfrak{c}(\mathbf{s}_i, \mathbf{s}_{i+1}), \tag{4}$$

where $\mathfrak{c}(\mathbf{s}_i, \mathbf{s}_{i+1})$[5] calculates the switching cost between the deployed solutions $\mathbf{s}_i$ and $\mathbf{s}_{i+1}$, and each deployed solution must be acceptable in all environments in which it remained operational, i.e., $\mathfrak{a}^{(b_i)}(\mathbf{s}_i) = 1, \mathfrak{a}^{(b_i+1)}(\mathbf{s}_i) = 1, \cdots, \mathfrak{a}^{(b_i+n_i)}(\mathbf{s}_i) = 1$. In (4), the fitness of a deployed solution in each environment can be interpreted as revenue, while the switching cost can be considered an outlay. Note that in each environment in which the previous deployed solution is reused, the switching cost is zero.

[4]Both quality threshold and $\mathfrak{a}^{(t)}(\cdot)$ are components of the problem and problem-specific.

[5]$\mathfrak{c}(\cdot)$ is part of the problem definition, i.e., it is problem-specific.

D. $\mathbf{ROOT^M_Q}$

As previously mentioned, in $\mathrm{ROOT^M}$ problems, multiple solutions are deployed concurrently. This class of ROOT problems arises in applications where there are multiple decision-makers with different preferences. For instance, in many cases, customers act as decision-makers, and the solutions deployed are lists of non-dominated services or products for the same purpose.

Among the various possible $\mathrm{ROOT^M}$ problems, only the multi-objective $\mathrm{ROOT^M_Q}$ has been studied in the existing literature [15–18], where the set of deployed solutions consists of Pareto-optimal solutions (POS). However, it is important to note that $\mathrm{ROOT^M}$ is not exclusively limited to multi-objective problems, and there are other classes of $\mathrm{ROOT^M}$ problems, such as multimodal ones [19], where the search space contains multiple moving global optima [20, 21].

In multi-objective $\mathrm{ROOT^M}$ problems, the deployed POS is retained until it becomes unacceptable according to a quality-based threshold. A multi-objective $\mathrm{ROOT^M}$ can be described as:

$$F^{(t)}\left(\mathbf{x}\right)=\left(f_1^{(t)}\left(\mathbf{x}\right),f_2^{(t)}\left(\mathbf{x}\right),\cdots,f_{\hat{m}}^{(t)}\left(\mathbf{x}\right)\right), \tag{5}$$

where F is a multi-objective problem, $f_i^{(t)}(\mathbf{x})=f\left(\alpha_i^{(t)},\mathbf{x}\right)$ is the ith dynamic objective function where $\alpha_i^{(t)}$ is a set of time-dependent control parameters of the ith objective function in the tth environment, and $\hat{m}$ is the number of objective functions. Given a multi-objective $\mathrm{ROOT^M_Q}$ problem $F(\mathbf{x},\alpha^{(t)})$ with $t_{\max}$ environments, the aim is to find a sequence of deployed sets of solutions $\widehat{\mathcal{P}}=\{\mathcal{P}_1,\mathcal{P}_2,\cdots,\mathcal{P}_{|\widehat{\mathcal{P}}|}\}$ where $\mathcal{P}_i$ is the ith deployed set of solutions, $|\widehat{\mathcal{P}}|$ is the total number of the deployed sets of solutions over $t_{\max}$ environments and $1\leq|\widehat{\mathcal{P}}|\leq t_{\max}$. Similar to $\mathrm{ROOT^S_Q}$, the main goal in solving $\mathrm{ROOT^M_Q}$ problems is to minimize the number of times a new set of solutions must be deployed, i.e., minimizing $|\widehat{\mathcal{P}}|$.

E. Discussion on ROOT problems

1) Comparing ROOT and other types of optimization problems: In the discussions provided in this part, we compare ROOT problems with other related classes of optimization problems including tracking moving optimum (TMO) in DOPs [10, 22–24], robust optimization [1, 25], and multi-scenario optimization [26, 27]. The aim of this section is to clarify the key differences between ROOT and other optimization problems, as well as to highlight situations where they may exhibit similar behaviors[6].

a) Comparing ROOT and TMO: Both ROOT and TMO problems involve dynamic environments [28] where deployed solutions change over time to react to environmental changes. An effective evolutionary dynamic optimization algorithm (EDOA) for TMO must address specific challenges, including how to deal with local and global diversity loss to maintain exploration and exploitation abilities over time, and how to react efficiently to environmental changes and quickly track desirable solutions in a new environment [10, 29]. To address these challenges, EDOAs (the state-of-the-art ones in particular) are complex algorithms that are constructed by assembling multiple components. ROOT problems typically share the challenges of TMO since, similar to a TMO algorithm, a ROOT method needs to handle optimization in dynamic environments [30]. Therefore, ROOT methods include components that are adapted from EDOAs [28].

To solve both ROOT and TMO problems, algorithms need to gather historical information, which is used for various purposes such as improving and accelerating the search process in the current environment [10]. This historical information is only useful if there are degrees of similarities and relations between successive environments [8], which typically depend on the magnitude of environmental changes [9]. In problems with low to moderate changes, where adequate degrees of similarities and relatedness exist, some aspects of the problems can be learned. For instance, some dynamical and morphological characteristics of promising regions can be identified and used in tracking the global optimum in TMO [10] and finding robust solutions in ROOT [30]. It is worth noting that as stochastic characteristics in environmental changes are prevalent in practice [9], what can be learned from historical information can typically be used for predicting some aspects of the problem, such as the approximate locations [31] or likelihood of reliability and robustness [32] of promising regions. However, due to the stochastic nature of environmental changes, the future fitness of solutions or the location of the next global optimum cannot be predicted with acceptable accuracy [9, 10]. In circumstances where there are no or a very low degree of similarity between successive environments, the historical information would be almost useless. Consequently, the mechanisms and components of the TMO and ROOT algorithms that work based on historical information would be ineffective, including those used in ROOT algorithms to predict the future robustness of solutions and promising regions [6, 32]. In such cases, which mostly happen in ROOT and TMO problems with abrupt environmental changes, i.e., those with huge spatial changes, perhaps the only option is to use restart methods, where the algorithm is restarted at the beginning of each environment [9]. Therefore, ROOT and TMO problems with severe changes share similar characteristics. However, note that such severe environmental changes are usually the result of cataclysmic events, such as failures in parts of the system. In many real-world DOPs, including ROOT and TMO problems, there are similarities and relatedness between successive environments, which can be learned and then used to solve these problems [9, 10, 33].

One main difference between ROOT and TMO is the notion of robustness over time in ROOT, which is not considered in TMO. In TMO, the focus is on finding a solution suitable for the current environment, while in ROOT, the focus is on finding and deploying a robust solution that may not be the best in the current environment but is acceptable for the current environment and will also remain acceptable for a number of future environments. In addition, in comparison to TMO problems, solving ROOT problems requires additional

[6]Note that to avoid distraction and simplify the presentation, the analysis and descriptions provided in this section are based on single-objective problems with single deployed solution at each point in time.

historical information, which is used to estimate the robustness of solutions and search space regions. Gathering, processing, and utilizing the obtained knowledge for robustness prediction and decision-making processes are challenging and make the ROOT algorithms more complex than TMO algorithms.

b) Comparing ROOT and multi-scenario optimization problems: Many real-world optimization problems aim to find a solution against a predefined set of scenarios [34]. Such problems, called multi-scenario optimization, consider a solution desirable only if its fitness values across the given scenarios are satisfactory [35, 36]. To tackle multi-scenario optimization problems, the objective values across the scenarios are usually aggregated [26], or different scenarios are considered as multiple objectives [37, 38].

At the first glance, ROOT problems may appear similar to multi-scenario optimization problems, where each of the current and future environments can be treated as a scenario. However, considering that the number of scenarios is predetermined and known in multi-scenario optimization, $\mathrm{ROOT_Q}$ and $\mathrm{ROOT_G}$ problems differ from multi-scenario optimization since the number of environments in which the deployed solution will be reused, i.e., the number of scenarios, is not known in these sub-classes of ROOT problems. In addition, the sequence of environments is important for $\mathrm{ROOT_Q}$ and $\mathrm{ROOT_G}$ problems, while the order of scenarios in multi-scenario optimization is usually not important.

In $\mathrm{ROOT_T}$ problems, however, the number of environments in which the deployed solution will be retained is available in the form of the size of the time window. Therefore, the size of the time window can be used as an equivalent of the number of scenarios in multi-scenario optimization. In addition, in $\mathrm{ROOT_T}$ problems, the goal is to maximize the average fitness value over each time window, which is not affected by the order of the environments/scenarios.

If we consider the current and $t_{\mathrm{win}}-1$ future environments in $\mathrm{ROOT_T}$ as a multi-scenario optimization problem with t_{win} scenarios, we may be able to use multi-scenario optimization algorithms to find a solution that is desirable over the t_{win} environments/scenarios. However, having knowledge of the $t_{\mathrm{win}}-1$ future environments (i.e., future objective functions) in $\mathrm{ROOT_T}$ is essential for this approach to be successful. One group of problems where future environments are known is reappearing/cyclic problems in which the landscape visits a finite set of environmental states repeatedly [33]. Therefore, under such special conditions, $\mathrm{ROOT_T}$ problems are similar to multi-scenario optimization. Note that when computational resources over t_{win} environments are sufficient (i.e., in problems with low change frequencies), a multi-scenario optimization algorithm can be used during the current time window to find a solution for deployment in the next time window.

In addition, certain $\mathrm{ROOT_T}$ problems with some *deterministic* dynamic characteristics [39], where accurately predicting future fitness values[7] of solutions is possible, might be translated into multi-scenario problems. However, it is important to note that for accurately predicting the future fitness values in these $\mathrm{ROOT_T}$ problems, we need sufficient samples in each environment to train accurate approximation and prediction models [6]. While in problems with regular and smooth morphological characteristics, we can train accurate models with smaller numbers of samples (i.e., solutions and their fitness values), in problems with rugged and irregular morphological characteristics, we need larger numbers of well-distributed samples for training accurate models [40, 41]. This will be more challenging in problems with high change frequencies, where the optimization algorithm is already struggling with a limited fitness evaluation budget in each environment to perform the search process. Consequently, using multi-scenario methods to solve such $\mathrm{ROOT_T}$ problems would be inefficient. Some existing ROOT methods use aggregation approaches employed in multi-scenario optimization and optimize multiple environments/scenarios simultaneously (see Section III-A1). Nevertheless, studies in [28, 32] showed that these methods are inefficient due to error-prone predictions and the lack of advanced dynamic handling components in the optimization algorithm's structure.

Finally, most real-world ROOT problems involve stochastic changes [9, 10] for which knowledge of future environments is unavailable. Predicting future fitness values of solutions and objective functions becomes highly error-prone in the presence of stochastic changes in such cases. Therefore, these problems cannot be solved using multi-scenario optimization, which requires knowledge of future environments beforehand. To overcome these challenges, a ROOT method is needed that can gather and process historical information to predict some aspects of the environment, such as the likelihood of reliability and robustness of promising regions, without attempting to predict the future fitness of solutions (i.e., future objective functions) [28].

c) Comparing ROOT and robust optimization problems: Robust optimization refers to a type of optimization that searches for a solution that is robust to various sources of uncertainty [25]. The desirable solution is usually deployed once, after completing the optimization process. In contrast, ROOT is the combination of robust optimization and optimization in dynamic environments that considers tracking, deploying, and switching robust solutions over time instead of just deploying a single solution in robust optimization [11, 42]. ROOT problems are typically ongoing optimization problems in dynamic environments, which means that the algorithm usually runs in an online manner[8]. Unlike robust optimization, in ROOT, the algorithm must respond to environmental changes and find desirable solutions based on the current and predicted future status of the problem.

In addition, uncertainties in robust optimization problems usually differ from those in ROOT problems, which arise due to environmental changes. In ROOT, the cumulative effect of these changes over time often results in significant uncertainties [9, 11], making it impossible to find a solution that remains robust to all environmental changes. Therefore, instead of searching for a single solution that can handle all

[7]By predicting the future fitness values of any solution in the search space, we predict the future environments.

[8]I.e., both the problem/system and the algorithm run simultaneously over time.

uncertainties, in ROOT, we must switch the deployed solution(s) over time as needed. Decisions about which solution(s) to deploy and when to switch the deployed solution(s) are made sequentially over time, and the uncertainty set is updated at each environmental state based on the relevant parameters and any new information that becomes available. This task becomes more challenging when uncertainties also change over time, such as in heterogeneous DOPs [10], where factors like the severity of changes, frequency of changes, and type of dynamics can all evolve over time.

Although ROOT and robust optimization problems differ in their characteristics and requirements, they can exhibit similar behavior in certain cases, such as in ROOT problems with both small change severity and insignificant cumulative environmental changes over time. However, it is important to note that not all ROOT problems with small change severity are similar to robust optimization problems. To illustrate this point, we provide an example in Fig. 2. In this example, we show the movement of the optimum positions in two cases over 500 environmental changes, with both cases having a small change severity of 0.1 and an initial optimum position of (0,0). As shown, while both cases have identical small change severity, the cumulative changes in case 1 are large, making robust optimization methods inapplicable to tackle it. By contrast, in case 2, the optimum position shifts randomly back-and-forth, resulting in a much smaller cumulative change over time compared with case 1. Thus, case 2 displays behavior similar to that of robust optimization problems because it is likely that there is a solution that remains acceptable across all environments. In general, any ROOT problem that has at least one solution whose performance remains desirable across all environments can be classified as a robust optimization problem. Nonetheless, many real-world problems do not have a solution that meets this criterion, i.e., there is no solution that is robust to all environmental changes. Thus, solutions deployed in such problems must be changed over time in response to environmental changes and to cope with the current status of the problem.

Furthermore, in the context of $\mathrm{ROOT_T}$, each time window can be considered as an individual robust optimization problem. However, since a solution must be chosen for deployment at the beginning of each time window, the robust optimization process for a specific time window needs to be conducted during the preceding time window. This approach necessitates prior knowledge of the future environments in the subsequent time window, which is typically unavailable. Moreover, as discussed in Section II-E1b, accurately predicting multiple future environments is usually impractical. Additionally, if we treat each time window as a distinct robust optimization problem and restart the optimization process in each time window, we would forfeit the opportunity to utilize historical information for enhancing the search process. Consequently, adopting an approach that treats each time window as an individual robust optimization problem may not be an effective strategy for tackling $\mathrm{ROOT_T}$ problems.

2) Discussions on requirements for changing or keeping solutions: Based on the definition of $\mathrm{ROOT_Q}$, the quality threshold and $\mathfrak{a}^{(t)}(\cdot)$ function determine the acceptability of

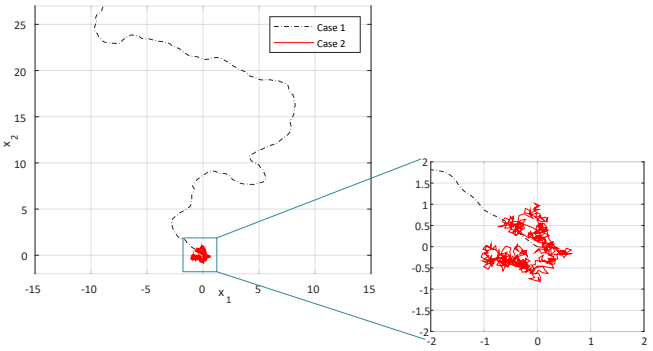


Fig. 2. Comparison of optimum position movements in problems with different cumulative changes over time. The figure shows the movement of optimum positions in two cases over 500 environmental changes, with both cases having a small change severity of 0.1 and an initial optimum position of (0,0). While case 1 exhibits large cumulative changes over time, case 2 shows random back-and-forth movements resulting in smaller cumulative changes.

the deployed solution(s) in the tth environment. Both the quality threshold and $\mathfrak{a}^{(t)}(\cdot)$ function are part of the problem definition and form a constraint that define the feasibility of the deployed solution(s). Depending on the requirements and characteristics of the problem/system, the quality threshold value can be fixed or varying over time. An example of $\mathrm{ROOT_Q}$ is crowd monitoring and management [43] which can be classified as a dynamic covering location problem [44]. In this problem, the desirable locations of field agent units change over time based on the current and predicted status of the dynamic crowd. However, in this problem, frequent relocation of field agent units is undesirable. To clarify the terms "undesirable" and "frequent changes" in this problem, assume that if the base station commands the field agent units to relocate every few seconds, the monitoring task is hindered as they are disturbed frequently. Therefore, in solving this problem, we seek solutions that remain acceptable for longer periods. A solution (e.g., the locations of the field agent units) in this problem is considered acceptable based on security and safety criteria. For instance, if the average distance between field agent units and people in the crowd exceeds a threshold, the location of the field agent units must be updated to a new position determined by the optimization algorithm. Note that in this specific application, the threshold value can change over time based on the current level of risk, such as the likelihood of incidents of crowd crushes or terrorism [45].

According to the definition of $\mathrm{ROOT_T}$ problems, there is a temporal constraint that prevents switching a deployed solution (or a set of solutions in $\mathrm{ROOT_T^M}$) before a specified end time. Various reasons may cause this temporal constraint in real-world problems, including:

- Insufficient resources to deploy a new solution until the end of the time window. For example, if changing the deployed solution requires specialized human resources who are only available every 24 hours, the current solution will have to remain in place until the next available shift.
- Time required to adapt to a new solution. For instance, in certain applications, users may need time to learn and

practice before they can fully benefit from a new solution.
- Completion of information gathering. In some scenarios, after deploying a new solution, some information need to be gathered before the solution can be switched. For example, in a commercial system, it may take time to gather market feedback on a newly produced product. Such information is usually necessary for decision-making about the next deployed solution.
- Impact of changes on other parts of the system. Changing certain aspects of a system in response to environmental changes may have negative consequences. For instance, investment plans provided to customers should remain fixed for a certain period (e.g., one year), despite changes in economic conditions, to avoid customer dissatisfaction and confusion.

Note that, in real-world $\mathrm{ROOT_T}$ problems, the size of the time window is usually finite, i.e., the problem-specific restrictions that define the time window during which we are unable to deploy a new solution are not permanent. Therefore, when the restrictions are lifted, i.e., when the current time window is finished, a new solution needs to be chosen for deployment to cope with the status of the problem/system during the new time window. In addition, the length of the time window may be fixed or may change over time, depending on the system and problem specifications and requirements.

Furthermore, in $\mathrm{ROOT_G}$ problems, the switching cost is a critical aspect of the problem that must be taken into account by the algorithm. This cost is problem-specific, which is defined as the practical expenses incurred during the preparation and implementation of a new solution, and it can vary depending on the specific requirements of the problem. The switching cost is a crucial factor that is used for determining whether to switch to a new solution or retain the existing one. For instance, if a more efficient product is designed, the costs associated with updating the production line and the projected future sales must be evaluated and compared to the revenue generated by the current product, i.e., the solution currently in use. This comparison helps determine whether the benefits of adopting the new solution outweigh the costs associated with the transition.

The degree of difference between the previously deployed solution and the new one can also affect the switching cost, with larger differences resulting in higher costs [8, 46]. For instance, longer relocations involving transportation and logistics require more fuel/energy and result in higher costs. In previous work, the parameters used to calculate the switching cost, such as fuel cost, were considered fixed over time. However, in practice, these parameters usually change over time, making it impracticable to treat them as fixed values.

Note that in problems where the switching cost is small, $\mathrm{ROOT^S_G}$ problems behave more like a TMO problem where the best-found solution in each environment is chosen for deployment [14]. On the other hand, in circumstances where the switching cost is very high, $\mathrm{ROOT^S_G}$ problems behave similarly to $\mathrm{ROOT^S_Q}$, i.e., due to the high switching cost, the current deployed solution is likely to be kept as long as it remains acceptable [28]. However, it is important to note that we cannot consider $\mathrm{ROOT^S_Q}$ as a special case of $\mathrm{ROOT^S_G}$ with high switching costs because, unlike $\mathrm{ROOT^S_G}$, many $\mathrm{ROOT^S_Q}$ problems do not consider switching costs as a primary factor to avoid frequent changes in the deployed solutions.

3) Limitations of Existing Research on ROOT Problems: While $\mathrm{ROOT^S_Q}$ has received significant attention, $\mathrm{ROOT^S_T}$ and $\mathrm{ROOT^S_G}$ have not been adequately investigated among $\mathrm{ROOT^S}$ problems. Additionally, out of the different types of $\mathrm{ROOT^M}$ problems, only multi-objective $\mathrm{ROOT^M_Q}$ problems have been explored, and no attention has been paid to multi-modal $\mathrm{ROOT^M_Q}$ or any other type of $\mathrm{ROOT^M_T}$ or $\mathrm{ROOT^M_G}$. Due to the prevalence of $\mathrm{ROOT_T}$ and $\mathrm{ROOT_G}$ problems in practice, there is a gap between real-world problems and academic research in the field.

Moreover, the majority of existing work in the field focus on problems with continuous search space. In addition, apart from the quality threshold, which can be considered a quality-based constraint, no other constraints, dynamic ones in particular, have been considered. It was also noted that all studies in the ROOT literature work on problems whose environmental changes occur only at discrete time steps, while the search space of many real-world problems continuously changes over time [47]. Finally, while the variable structure of many real-world problems is partially separable [48, 49], the majority of studies, particularly those focusing on $\mathrm{ROOT^S}$ problems, have focused on fully non-separable problems.

It should also be noted that not all multi-objective ROOT problems are $\mathrm{ROOT^M}$. In fact, multi-objective ROOT problems can be classified as either $\mathrm{ROOT^M}$ or $\mathrm{ROOT^S}$ based on the number of deployed solutions. There are usually *several* decision-makers with different preferences in multi-objective $\mathrm{ROOT^M}$ problems. In such problems, the main aim is to seek a robust POS. By providing and implementing the POS, each decision-maker can pick a solution for deployment according to his/her own preferences. In multi-objective $\mathrm{ROOT^S}$ problems, on the other hand, there is one deployed solution at each point in time, which is chosen by a solo decision-maker whose preferences may change over time.

III. ROOT METHODS

In this section, we review the methods developed for solving ROOT problems. To have a better understanding of how these methods work and their advantages and disadvantages for solving different classes of ROOT problems, instead of classifying them according to the type of problems, we propose a taxonomy for these methods based on the approaches applied to finding robust solutions. This taxonomy also allows us to analyze these approaches independent of the type of ROOT problems since some of them can be used for multiple classes of ROOT problems. In this taxonomy, these approaches are classified into:

1) Approaches that use the predicted fitness of the candidate solutions to predict their robustness.
2) Approaches that work based on the estimated robustness of the promising regions instead of the robustness of the candidate solutions.
3) Approaches that take additional objective functions into account for finding robust solutions.

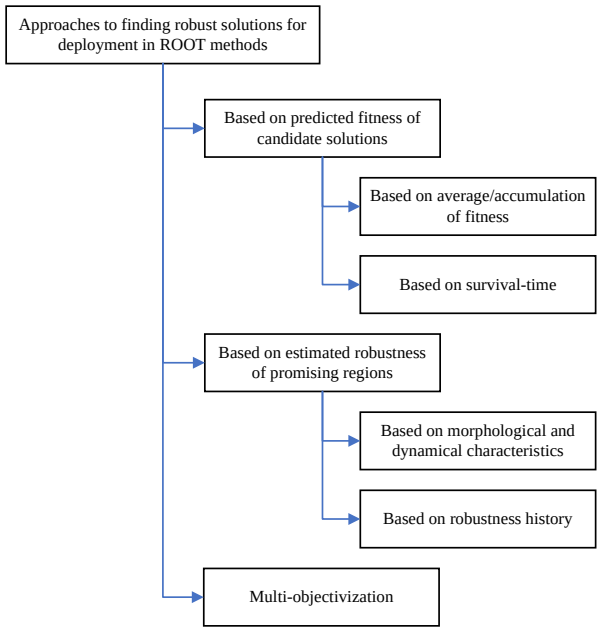


Fig. 3. Taxonomy of the ROOT methods.

The proposed taxonomy for ROOT methods is illustrated in Fig. 3. In the following, we provide an in-dept technical review[9] of the developed ROOT methods according to the proposed taxonomy.

A. Finding robust solutions based on predicted fitness of candidate solutions

These methods use historical data to predict the fitness values of candidate solutions in the forthcoming environments. The robustness of a candidate solution is then determined based on its predicted fitness values. In these ROOT methods, the main objective function is replaced by a *substitute* objective function to find robust solutions. This class is further classified into: *fitness accumulation/average*-based, and *survival-time*-based methods.

In the fitness accumulation/average-based methods, the search space is constructed based on the accumulation/average of the solutions' fitness values in the current environment, a predefined number of the previous environments (which are constructed by the historical data and approximation component [50]), and a predefined number of the future environments [6, 12] (constructed by prediction component). On the other hand, in the survival-time-based approaches, the search space is constructed based on the survival time of solutions, which is normally the number of successive upcoming environments that a solution remains acceptable, determined based on the predicted fitness values of the candidate solutions and acceptability criteria. In the following, we describe these methods.

[9]Since we aim to make this survey self-contained for readers who may be new to the field or have limited knowledge of the topic, and because we aim to do a deep analysis of the main methods in the field, our review includes comprehensive details and descriptions.

1) Fitness accumulation/average-based approach: Jin et al. [6] proposed the first fitness-accumulation-based framework that searches for robust solutions by means of fitness approximation and prediction. The goal of this method, which essentially tackles $\text{ROOT}^{\text{S}}_{\text{Q}}$ problems, is to find solutions that remain acceptable after several environmental changes. This method uses the predicted fitness values of the candidate solutions in a predefined number of forthcoming environments to evaluate their robustness. The main components of this method include a population-based optimizer, an archive, an approximator (also known as a surrogate model [51]), and a predictor. The optimizer has two main responsibilities which are gathering data for training the predictor and performing optimization. In each iteration, all candidate solutions of the optimizer, their fitness values, and the associated environment number are archived in the archive. The archived information are then used to train a prediction method to estimate the future fitness values of the candidate solutions. More specifically, for a candidate solution, a time-series of its fitness values over a predefined previous environments plus the current environment are used to train a predictor for estimating its future fitness values in the upcoming q environments. Since the past fitness values of a solution may not be in the archive, an approximator is used to estimate these past fitness values. The aforementioned components cooperate through the following metric:

$$\breve{f}^{(t)}(\mathbf{x}) = \sum_{i=t-p}^{t-1} f'^{(i)}(\mathbf{x}) + f^{(t)}(\mathbf{x}) + \sum_{j=t+1}^{t+q} f''^{(j)}(\mathbf{x}), \quad (6)$$

where $\breve{f}$ is the substitute objective function, t is the index of the current environment, f' is the approximation function where $f'^{(t-j)}(\mathbf{x})$ estimates the fitness value of $\mathbf{x}$ in the $(t-j)$th environment, $f^{(t)}$ is the actual objective function in the current environment t, f'' is the prediction function where $f''^{(t+j)}(\mathbf{x})$ estimates the fitness value of $\mathbf{x}$ in the $(t+j)$th environment, and p and q are the numbers of considered past and future environments, respectively. In this method, a particle swarm optimization (PSO) [52] that randomizes a predefined portion of the population after each environmental change [53] is used as the optimization component, a radial basis function is used as the approximator [40], and autoregression is used as the predictor. In [54], the impact of different predictor methods on the performance of the framework in [6] is investigated. It is shown in [54] that the results are not significantly different when more advanced predictors are used.

As shown in (6), the substitute objective function considers p past approximated fitness values of a candidate solution. However, robustness is defined based on both current and future fitness values. Moreover, the past approximated fitness values are implicitly taken into account when training the predictor. Therefore, further studies are needed to investigate the necessity and effectiveness of including past approximated fitness values in (6).

As stated before, the method in [6] is used to tackle $\text{ROOT}^{\text{S}}_{\text{Q}}$ problems. This method keeps the deployed solution for as long as it remains acceptable and tries to maximize the average survival time of the deployed solutions. However,

according to the substitute objective function in (6), this method does not consider the *acceptability* of solutions in the current and upcoming environments. In other words, only the accumulation of the fitness values of a candidate solution from p past environments to q future environments are taken into account in (6). For example, an *unacceptable* solution in the current environment with high predicted future fitness values can be chosen for deployment if its accumulated fitness value is the highest. However, such a solution is infeasible for deployment due to its unacceptable current fitness value. Since the method in [6] does not consider current and future acceptability of candidate solutions and focuses on the fitness values, it is more suitable for solving $\mathrm{ROOT}_{\mathrm{T}}^{\mathrm{S}}$ problems.

Later, Fu et al. [13] proposed a method for tackling $\mathrm{ROOT}_{\mathrm{T}}^{\mathrm{S}}$ problems by modifying the method in [6]. With the exception of the substitute objective function, all the components of the method in [6] are also used in the $\mathrm{ROOT}_{\mathrm{T}}^{\mathrm{S}}$ method in [13]. For tackling $\mathrm{ROOT}_{\mathrm{T}}^{\mathrm{S}}$ problems, this method tries to find a solution for deployment at the first environment of each time window to maximize the average fitness over the time window. The following substitute objective function has been used in this method:

$$\breve{f}^{(t)}(\mathbf{x}) = \frac{1}{t_{\mathrm{win}}}\left(f^{(t)}(\mathbf{x}) + \sum_{j=t+1}^{t+t_{\mathrm{win}}-1} f''^{(j)}(\mathbf{x})\right). \quad (7)$$

As can be seen, the substitute objective function in (7) is similar to that of the method in [6] when we set $p = 0$ and $q = t_{\mathrm{win}} - 1$ in (6).

In [55, 56], the performance of the $\mathrm{ROOT}_{\mathrm{T}}^{\mathrm{S}}$ method in [13] is investigated where different differential evolution (DE) algorithms [57] are used as the optimization component. In addition, the impact of different approximation components on the performance of the $\mathrm{ROOT}_{\mathrm{T}}^{\mathrm{S}}$ method in [13] is investigated in [58]. It is shown in this paper that more advanced approximation methods can improve the performance of the $\mathrm{ROOT}_{\mathrm{T}}^{\mathrm{S}}$ method in [13]. However, since the empirical study has been conducted on a limited set of 2-dimensional problems, the impact of different approximation methods on problem instances with higher number of dimensions is still unclear.

Fitness accumulation/average-based approaches have also been used to find multiple solutions for deployment simultaneously. In [15], a framework for solving multi-objective $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{M}}$ problems was introduced where the goal is to find a robust POS for deployment. A robust POS may not be the true POS in an environment, but it remains relatively close to the true POS for at least two successive environments. To solve multi-objective $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{M}}$ problems, the problem shown in (5) is transformed into a substitute one based on the multi-objective robust solution definitions introduced by Deb and Gupta in [59]:

$$\breve{F}^{(t)}(\mathbf{x}) = \left(\breve{f}_1^{(t)}(\mathbf{x}), \breve{f}_2^{(t)}(\mathbf{x}), \cdots, \breve{f}_{\hat{m}}^{(t)}(\mathbf{x})\right), \quad (8)$$

where the ith substitute objective function $\breve{f}_i^{(t)}(\mathbf{x})$ is:

$$\breve{f}_i^{(t)}(\mathbf{x}) = \frac{1}{q}\left(f_i^{(t)}(\mathbf{x}) + \sum_{j=t+1}^{t+q-1} f_i''^{(j)}(\mathbf{x})\right), \quad (9)$$

where $f_i^{(t)}$ is the actual ith objective function in the current environment t, f'' is a prediction function where $f_i''^{(t+j)}(\mathbf{x})$ estimates the ith fitness value of $\mathbf{x}$ in $(t+j)$th environment, and $q-1$ defines the numbers of considered future environments. According to (9), this method works based on the predicted fitness of candidate solutions in the upcoming environments. Similar to the method in [6], an archive and an approximation component are used for each objective function to provide data for training the predictors. Besides, an evolutionary multi-objective optimization method is used in this framework (MOEA/D [60] is used in [15]). This framework is also used in [16–18, 61] for solving $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{M}}$ problems. In [16], ensemble prediction methods are used in this framework to improve the prediction accuracy. In [17] and [61], a grid-based multi-objective brain storming algorithm [62] with hybrid mutation operation and a modified version of non-dominated sorting genetic algorithm [63] were used as the optimization component in this framework, respectively. In [18], an alternative prediction component is used to estimate the future fitness values of the candidate solutions. In this method, instead of using time series of previous fitness values of candidate solutions for predicting their future fitness values, future environmental parameters (i.e., α-variables) are predicted. Then, using the predicted environmental parameters and the baseline functions, the future fitness values of candidate solutions are estimated and then used in (9).

2) Survival-time-based approach: Fu et al. [12, 13, 42] proposed a survival-time-based framework to tackle $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{S}}$ problems. The main components of this framework are the same as the method in [6], however, the substitute objective function is different:

$$\breve{f}^{(t)}(\mathbf{x}) = \begin{cases} 0, & \text{if } f^{(t)}(\mathbf{x}) < \mu \\ 1 + \max\{n \mid \forall i \in \{1,2,\cdots,n\} : f''^{(t+i)}(\mathbf{x}) \geq \mu\}, & \text{otherwise} \end{cases}, \quad (10)$$

where μ is the quality threshold. According to (10), if the current fitness value (i.e., in tth environment) of a candidate solution is less than μ, it is considered as a non-robust solution. On the other hand, for a candidate solution whose fitness value in the current environment is acceptable, its robustness value is the number of forthcoming successive environments in which its fitness values are predicted to remain above μ. Using (10), the $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{S}}$ method in [13] searches for solutions with higher estimated robustness since future acceptability of the candidate solutions are taken into account in the substitute objective function. In [64], the performance of the $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{S}}$ method in [13] is investigated where DE is used as the optimization component.

Since (10) considers the robustness of candidate solutions, it seems to be a more suitable substitute objective function to tackle $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{S}}$ in comparison to (6). However, (10) suffers from a significant issue which is best explained by several illustrative examples. Figure 4 illustrates search spaces

generated by (10) using different quality threshold values $\mu = \{40, 45, 50\}$. As can be seen, the search spaces generated by (10) contain vast plateau regions and a few narrow peaks. Moreover, Figs. 4(b), 4(c), and 4(d) show that by increasing the quality threshold value, the regions containing robust solutions shrink, and the plateau regions expand. Such landscapes are extremely challenging for the optimizers due to the stagnation and premature convergence issues [65]. These challenges are intensified by increasing the problem dimensionality. It is shown in [30] that PSO fails to find any promising region in a 10-dimensional problem generated by (10) where $\mu = 40$.

B. Finding robust solutions based on estimated robustness of promising regions

Yazdani et al. [28] proposed a class of ROOT methods that focused on the robustness of promising regions instead of candidate solutions. In addition, this class operates on the landscape generated by the actual objective function instead of searching for robust solutions in the search space constructed by a substitute objective function. These methods rely on advanced components of multi-population evolutionary dynamic optimization algorithms to handle the challenges caused by environmental changes, locate and track multiple moving promising regions over time, monitor these promising regions, and gather some information about them, which is used to estimate the likelihood of reliability and robustness of each covered promising region. We further classify these ROOT methods into those that consider the morphological and dynamical characteristics of promising regions and those that rely on the robustness history of promising regions.

1) Considering the morphological and dynamical characteristics of promising regions: Yazdani et al. [68] proposed a $\text{ROOT}^{\text{S}}_{\text{Q}}$ framework which estimates the robustness of the promising regions based on some *reliability* factors. In this framework, the robustness of the promising regions is estimated based on some information gathered by the optimization component. Unlike the ROOT methods described in Section III-A that operate on the search space of a substitute objective function to find robust solutions, this framework works on the search space of the *actual* objective function. The components of this framework include a multi-population EDOA capable of locating and tracking multiple moving promising regions, an archive for each sub-population that is used for storing some historical information about each covered promising region, and a decision-maker that chooses solutions for deployment. Thus, this framework does not rely on any approximated and/or predicted fitness values of candidate solutions.

Fitness fluctuation degrees after each environmental change is one of the main pieces of information gathered in [68] about the moving promising regions. To estimate the fitness fluctuation of each promising region that is covered by the ith sub-population, the following calculation is done after each environmental change:

$$\tau_i^{(t)} = \left| f^{(t-1)}\left(\mathbf{g}_i^{*(t-1)}\right) - f^{(t)}\left(\mathbf{g}_i^{*(t-1)}\right) \right|, \qquad (11)$$

where $\tau_i^{(t)}$ is the fitness fluctuation at the tth environmental change and $\mathbf{g}_i^{*(t-1)}$ is the best-found position by the ith sub-population in the $t-1$th environment. τ_i values are stored in the ith sub-population's archive. Thereafter, the average values of τ_i in the past environments ($\bar{\tau}_i$) is used for estimating the robustness of the covered promising region by the ith sub-population using:

$$\mathfrak{r}\left(\mathbf{g}_i^{*(t)}, \bar{\tau}_i\right) = \begin{cases} 1, & \text{if } f^{(t)}\left(\mathbf{g}_i^{*(t)}\right) - \bar{\tau}_i \geq \mu \\ 0, & \text{otherwise} \end{cases}. \qquad (12)$$

If $\mathfrak{r}\left(\mathbf{g}_i^{*(t)}, \bar{\tau}_i\right) = 1$, it means that $\mathbf{g}_i^{*(t)}$ is likely robust to at least one upcoming environmental change, i.e., it is expected that its next fitness value will remain above the threshold μ. In the tth environment, if a new solution must be chosen for deployment, the algorithm first forms a set $\mathcal{C}$ that contains the robust promising regions (i.e., $\left\{ i \in \mathcal{C} | \mathfrak{r}\left(\mathbf{g}_i^{*(t)}, \bar{\tau}_i\right) = 1 \right\}$). In the next step, a decision-making process is used to choose a promising region from $\mathcal{C}$ based on a predefined strategy. In [68], the following strategy is used:

$$c^{*(t)} = \underset{i \in \mathcal{C}}{\operatorname{argmax}} \left(f^{(t)}\left(\mathbf{g}_i^{*(t)}\right) - \bar{\tau}_i \right), \qquad (13)$$

where $c^{*(t)}$ is the chosen promising region in the tth environment which is the one from $\mathcal{C}$ whose best-found position has the highest worst-estimated future fitness value. Next, the best-found position in the promising region $c^{*(t)}$, which is likely near the summit position, is chosen for deployment. The reason behind choosing the best-found position is that robust solutions usually exist around the summits of the robust promising regions [69, 70].

Some other strategies that work based on the estimated shift and height severity values are proposed in [30]. In this approach, some additional information is gathered and archived by sub-populations to estimate shift and height severity values. In each pair of successive environments t and $t+1$, the following values are calculated and archived for the promising region covered by the ith sub-population:

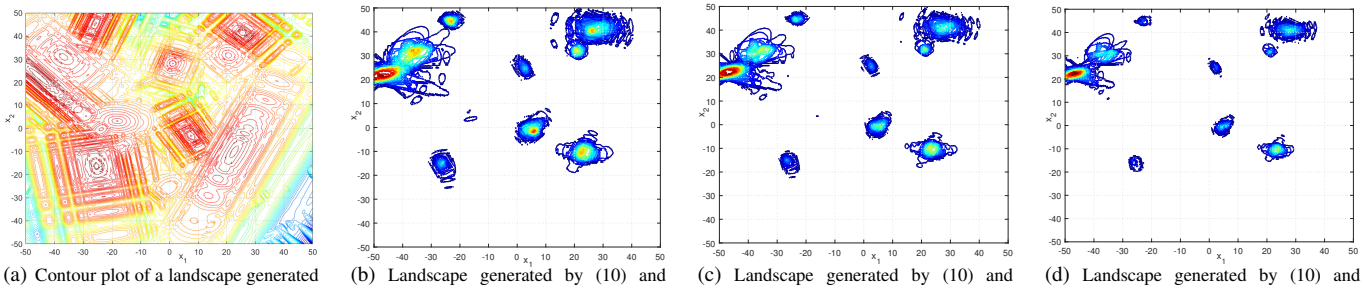
$$s'^{(t)}{}_i = \left\| \mathbf{g}_i^{*(t)} - \mathbf{g}_i^{*(t+1)} \right\| \qquad (14)$$

and

$$h'^{(t)}{}_i = \left| f^{(t)}\left(\mathbf{g}_i^{*(t)}\right) - f^{(t+1)}\left(\mathbf{g}_i^{*(t+1)}\right) \right|, \qquad (15)$$

where $s'^{(t)}{}_i$ and $h'^{(t)}{}_i$ are estimated relocation length and height difference of the best-found position by the ith sub-population in the tth and $t+1$th environments. For each sub-population i, the *average* values of archived $s'^{(t)}{}_i$ and $h'^{(t)}{}_i$ show the estimated shift ($\bar{s}'_i$) and height ($\bar{h}'_i$) severity values of its covered promising region, respectively.

The idea behind using estimated shift and height severity values in these strategies is that choosing the summit positions of the promising regions with smaller shift and/or height severity values for deployment can be more robust to environmental changes. More specifically, smaller shift severity values in a promising region during environmental changes result in smaller fitness drops when the summit position is



(a) Contour plot of a landscape generated by GMPB. (b) Landscape generated by (10) and μ=40. (c) Landscape generated by (10) and μ=45. (d) Landscape generated by (10) and μ=50.

Fig. 4. (a) shows the contour plot of the actual current landscape made using the Generalized Moving Peaks Benchmark (GMPB) [66, 67] with randomly generated 10 promising regions. (b)-(d) show the generated search space by (10) using the current environment in (a) and its future environments, where μ is set to 40, 45, and 50, respectively. Note that actual future fitness values are used to generate (b)-(d) instead of the predicted values. This is done to remove the prediction error impact on the constructed search spaces shown in these figures.

chosen for deployment. On the other hand, larger shift severity values lead to more severe fitness drops, which result in lower robustness. Moreover, promising regions with smaller height severity values are unlikely to suffer from significant fitness drops caused by decreased height values. On the other hand, the promising regions with larger height severity degrees may suffer from large fitness drops after environmental changes that result in lower robustness. Therefore, if the best-found position in a promising region with smaller shift and/or height severity values is chosen for deployment, its likelihood of remaining acceptable after upcoming environmental changes will be higher.

The following three strategies were presented in [30] based on the estimated shift and height severity values of the promising regions in $\mathcal{C}$:

$$c^{*(t)} = \underset{i \in \mathcal{C}}{\operatorname{argmin}} \left(\bar{s}'_i\right), \tag{16}$$

$$c^{*(t)} = \underset{i \in \mathcal{C}}{\operatorname{argmin}} \left(\bar{h}'_i\right), \tag{17}$$

and

$$c^{*(t)} = \underset{i \in \mathcal{C}}{\operatorname{argmin}} \left(\frac{\bar{s}'_i}{\bar{s}'_{\max}} + \frac{\bar{h}'_i}{\bar{h}'_{\max}}\right), \tag{18}$$

where $\bar{s}'_{\max}$ and $\bar{h}'_{\max}$ are the largest $\bar{s}'$ and $\bar{h}'$ values among promising regions in $\mathcal{C}$, respectively. The experimental results in [28, 30] demonstrate that solutions chosen by (18), which considers both shift and height severity values, have longer survival times in comparison to the ones chosen by (16) and (17). Finally, in [14], a strategy is used that works based on the switching cost between the previous deployed solution $\mathbf{s}$ and the best-found positions in the promising regions in $\mathcal{C}$:

$$c^{*(t)} = \underset{i \in \mathcal{C}}{\operatorname{argmin}} \left(\mathfrak{c}\left(\mathbf{s}, \mathbf{g}_i^{*(t)}\right)\right), \tag{19}$$

where $\mathfrak{c}(\mathbf{s}, \mathbf{x})$ calculates the switching cost between the deployed solution $\mathbf{s}$ and $\mathbf{x}$. In [14], Euclidean distance between the deployed solution and $\mathbf{x}$ is used as the switching cost.

2) Considering the history of the robustness of promising regions: One useful piece of information about each promising region is its robustness history, which can be used in the decision-making process for choosing a promising region from which a solution is chosen for deployment. A ROOT framework is proposed in [32], which, like the proposed framework in [30], includes a decision-maker, a multi-population EDOA for finding and tracking moving promising regions, and an archive for each sub-population. The main differences between these two frameworks are the archived information and the decision-making process. In [32], the historical knowledge stored in each archive includes the best-found solutions in the covered promising region in previous environments, as well as the environment number associated with each solution. The degree of robustness of each covered promising region is estimated in this framework after each environmental change. To this end, the archived solutions are re-evaluated after each environmental change, and their acceptability is determined using (S-12) in the current environment. The estimated robustness degree shows the number of successive environments for which, if any of the previous best-found solutions in the promising region were deployed, they would have continued to remain acceptable up until the current environment. It is worth mentioning that efficiently managing the archives and controlling the computational burden of the robustness estimation process in this framework are challenging and complex tasks that involve several additional mechanisms.

After determining the estimated robustness degree for all covered promising regions, a decision-maker is responsible for selecting one promising region from which the next solution for deployment is chosen. In [32], the decision-maker simply selects the promising region with the highest estimated robustness degree. It is shown in [32] that the promising regions with larger values of the estimated robustness have some suitable morphological and dynamical characteristics that increase the likelihood of their robustness to the upcoming environmental changes.

C. Multi-objectivization

In these methods, while the problem is single-objective by nature, some additional objectives are also considered to find a desirable solution for deployment. Since these multi-objective methods are developed for solving single-objective ROOT problems, they usually need to choose one solution for deployment from the POS.

A double layered bi-objective $\mathrm{ROOT}^{\mathrm{S}}$ method is proposed in [71]. In the first layer, the algorithm tries to find POS based on the substitute objective functions in (7) and (10) for each environment. Therefore, this algorithm tries to tackle $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$ and $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{T}}$ simultaneously. Afterwards, in the second layer, the algorithm tries to find the best sequence of robust solutions based on the found POSs over different environments in the first layer. Such an approach for finding POSs in the first step and then selecting a sequence of solutions from them is more suitable for problems in which the sequence of environments recurs [9, 72]. Therefore, the set of solutions chosen in the second layer of this method can be reused in the future reappearing environments. The concept of multi-objectivization by simultaneously considering $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$ and $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{T}}$ is also used in [61].

In [73], a multi-objective approach called $\mathrm{ROOT/SC}$ is proposed for optimizing two objectives, including survival time maximization in (10) and switching cost minimization. The switching cost was defined as the Euclidean distance between the deployed solution and a candidate solution. The algorithm keeps the deployed solution until it becomes unacceptable. When a previously deployed solution becomes unacceptable, a new solution must be chosen from the POS for deployment. From the aspect of requirements for changing or keeping the deployed solution, this method can be considered a $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$ method. However, from the aspect of choosing a new solution for deployment, depending on the defined preferences, this method can also be considered as $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{G}}$ since it tries to minimize the switching cost.

Note that since $\mathrm{ROOT/SC}$ uses (10) as its $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$ objective, it also suffers from the difficulty of the search space generated by this objective function in higher dimensions (see Fig. 4). An improved version of $\mathrm{ROOT/SC}$, called $\mathrm{ROOT/SCII}$, is proposed in [74]. In this algorithm, a helper objective, which is the actual objective function, is also considered, which helps in ameliorating the difficulty caused by (10) in higher dimensions.

D. Discussion on ROOT methods

1) Optimization techniques used in ROOT methods: ROOT methods are complex algorithms composed of multiple components, including those related to change reactions, information gathering, prediction, and decision-making. One important component of ROOT methods is the optimization component.

The majority of the work in the field utilizes evolutionary algorithms (EAs) as the optimization engine in ROOT methods. Using EAs in ROOT methods has several advantages, including:

- No prior knowledge required: EAs do not require any prior knowledge about the problem being solved. This is particularly advantageous in solving ROOT problems, where the underlying dynamics of the problem may be unknown or changing over time.
- Flexibility: EAs are highly flexible and can be easily adapted to different types of optimization problems, including constrained, multi-objective, discrete, and continuous. They are also able to handle non-linear and non-convex optimization problems, which can be challenging for other optimization methods. In addition, EAs can be used in conjunction with controller components, which can control EAs to adapt to various situations over time, such as change reaction components that prepare EAs for new environments.
- Effective information gathering: EAs are mostly population-based algorithms, allowing them to explore vast areas of the search space, which is beneficial for gathering information needed for predicting robustness.
- Multi-population: EAs can be easily parallelized into multiple populations using population division and management components [10], allowing them to cover various regions of the landscape in order to improve information gathering and quicker reactions to environmental changes.

PSO and DE are the most commonly used EA approaches in ROOT methods. Several studies have shown that EAs are effective in addressing ROOT problems, including those with complex morphological and dynamical characteristics [32, 66]. However, it is important to note that there may be limitations or drawbacks to using EAs in ROOT algorithms, such as high computational costs or difficulty in handling large-scale problems [48, 49]. To the best of our knowledge, random sampling [69] is the only non-EA optimization technique used in ROOT methods. However, its effectiveness is limited to low-dimensional problems.

2) EDOAs' components used in ROOT algorithms: The majority of challenges associated with TMO in DOPs can also be observed in ROOT problems. Similar to TMO approaches, all ROOT methods also need to overcome challenges such as global and local diversity loss and limited computational resources in each environment [10]. Almost all ROOT methods use the components of EDOAs that are designed to address these challenges. In fact, a ROOT method is constructed by combining components of EDOAs with explicit archives and some specific components for finding robust solutions, such as the approximation and prediction components in [6, 13] and the decision-maker component in [30]. The performance of ROOT methods is highly affected by the effectiveness of the EDOA used in these methods. Consequently, using a less effective EDOA deteriorates the performance of the ROOT methods.

For example, in [6, 13], a very simple single-population restart-based PSO (RPSO) from [53, 75] is used. It is shown in [76–78] that this EDOA is ineffective in optimizing many DOPs. This inefficiency of RPSO is not surprising, as this EDOA is constructed by only integrating a simple change-reaction-based global diversity control component, which randomizes a predefined portion of the inferior individuals after environmental changes, to a single population PSO[10]. By only using a simple global diversity control component, RPSO is not capable of performing adequate exploration and exploitation in the new environments and addressing the specific challenges of DOPs. As RPSO suffers from several shortcomings, it negatively impacts the performance of ROOT

[10]RPSO also has a change detection component, which is usually redundant as in most real-world DOPs, the algorithms are informed about environmental changes by other parts of the system, such as sensors [28].

methods in [6, 13]. Therefore, in designing ROOT methods, we must consider that the role of the used EDOA is crucial and that using more efficient EDOAs results in better performance. It is worth mentioning that to further improve the efficiency of ROOT methods, instead of using generic EDOAs that are particularly developed for TMO, researchers should pay more attention to developing EDOAs specifically designed for ROOT. For example, in most computational resource allocation methods used in EDOAs, the objective function values of solutions, in particular that of the best individual, are considered, which is suitable for TMO. However, in solving ROOT problems, instead of objective function values, robustness of solutions and promising regions should be taken into account in the computational resource allocation components [32].

3) Approximation component: In this part, we provide a discussion of the approximation component used for providing previous approximated fitness values in the ROOT methods that work based on the estimated future fitness of candidate solutions. In [6, 12, 13, 15], in the tth environment, a surrogate models generated by the approximation component are used for approximating $(t-a)$th to $(t-1)$th environments. The $(t-j)$th surrogate model ($j \in \{1, \cdots, a\}$) is responsible for approximating the fitness values of candidate solutions in the $(t-j)$th environment. Using these surrogate models, the previous fitness values of a candidate solution are estimated to form a time series for training the prediction component.

To have more accurate surrogate models, it is crucial to have sufficient samples with adequate distribution stored in the archive for each environment. EDOAs' candidate solutions over the optimization process are the main source of the archived samples. However, since the EDOAs quickly converge to the promising regions in each environment, the provided samples are not always well-distributed to build an accurate surrogate model. As a result, there are usually areas without adequate samples. To address this issue, a specific hypercube design [79] is used in [6] for initializing a predefined portion of the population after each environmental change. However, systematically initializing the population in each environment will also become challenging for problems with larger numbers of dimensions or search ranges. The reason is that the search space can become overly large in comparison to the population size that even systematically distributing individuals cannot properly provide the necessary samples to build an accurate surrogate model. One way to ameliorate this problem is that, besides the EDOA's candidate solutions, the algorithm also systematically evaluates some additional solutions to improve the surrogate modeling in each environment. This way, however, may be prohibitive or ineffective when the available computational resources are limited in each environment (e.g., problems with high change frequency). This ineffectiveness would be a consequence of the additional usage of the computational resources (i.e., fitness evaluation burden) by the sampling process, while the optimization component is usually struggling with the shortage of the available computational resources to perform sufficient exploration/exploitation [78].

The effectiveness of an approximation component is vital as it can negatively impact the effectiveness of the prediction component. Considering the aforementioned issues and challenges in using the approximation component, careful considerations should be made to decide if and how this component, at least in its current form, should be used. The approximation component for ROOT methods is first used in [6] since it is assumed that the algorithm in the tth environment has no access to $f^{(t-j)}$ for evaluating the fitness of a candidate solution in the $(t-j)$th environment. However, this assumption is not true for many real-world problems, because in many of these problems, access to the previous objective functions is available. For example, while the baseline function in the problem shown in (1) does not usually change over time, the environmental changes are actually the result of changes in the environmental parameters α. In many real-world DOPs and ROOT problems, access to the previous environmental parameters $\alpha^{(t-j)}$, such as previous costs, temperature, pressure, speed, available resources, positions, the number of customers, and lists of tasks, is possible. As a result, by using the baseline function and the previous environmental parameters, we can evaluate the previous fitness values of a candidate solution. Consequently, in cases like the above, it may not be necessary to use the approximation component if the available computational resources in each environment suffice to also evaluate previous fitness values. Furthermore, in cases where we need to use the approximation component due to limited computational resources, using the previous environmental parameters, we can improve the accuracy of the surrogate models in the current environment by providing additional samples from important areas (e.g., in areas containing high quality robust solutions).

To address the challenges of using the approximation component in providing the time series data for predicting the future fitness values of candidate solutions, α-variables (i.e., environmental parameters) are predicted in [18]. In this approach, using the baseline function $f(\cdot)$ and predicted α-variables, the fitness values of candidate solutions in future environments are calculated. Therefore, by using this approach, we no longer need an approximation component. However, the computational burden of this approach is huge because it performs fitness evaluations in order to calculate the predicted fitness values. In fact, this prediction approach consumes at least 50% of the fitness evaluations.

4) Error of future fitness prediction: In this part, we discuss the error of prediction component, which is used in the ROOT methods that work based on the predicted fitness values of candidate solutions. To train a predictor for estimating the future fitness of a candidate solution, the current and approximated previous fitness values of the candidate solution are used as the training data. As stated in Section III-D3, using approximation methods for estimating the previous fitness values of a candidate solution in its current form is error-prone and sometimes impractical. In addition, as shown in [28], even if the true previous fitness values of solutions are used for training the predictor, using prediction to estimate the future fitness values of candidate solutions is still error-prone. Figure 5 depicts the error of predicting the future fitness values of solutions in three consecutive environments with random

dynamics. Although the predictors are trained using the true previous fitness values, the error of prediction is significant and grows even larger when we extrapolate further into future environments, as shown in Figs. 5(d), 5(e), and 5(f). Note that the prediction error will be even larger when the approximated previous fitness values are used instead of the true previous fitness values for training.

The scale of the prediction error in methods that work based on the predicted fitness values of solutions depends heavily on the type of dynamics. In problems whose environmental changes pose some regular and deterministic characteristics, such as circular or linear dynamics [80, 81], these methods can be effective. However, in problems with stochastic environmental changes, the error in the prediction models can be significant, hindering the methods from finding robust solutions.

5) Challenges in predicting robustness of promising regions: As described in Section III-B, for predicting the robustness of the promising regions, the methods/strategies use some historical knowledge, such as the average fitness fluctuation, shift severity, height change severity, and robustness. Using the aforementioned historical knowledge, the heuristic approaches used in these methods try to determine the promising regions that are *likely* (i.e., not *guaranteed*) robust to the upcoming environmental changes. In fact, the heuristic approaches used do not guarantee the future robustness of solutions chosen for deployment from the promising regions in $\mathcal{C}$. As a matter of fact, the future robustness of solutions cannot be generally guaranteed by any other approach. It is worth mentioning that these heuristics run the risk of losing some actual robust promising regions whose $\mathfrak{r}(\cdot) = 0$ [30] or have smaller estimated robustness degrees [32]. However, when $\mathfrak{r}(\cdot) = 0$ for a promising region or its estimated robustness degree is small, both of which are considered negative observations, the likelihood of its robustness decreases significantly. From a Bayesian perspective, such negative observations make the robustness of the promising region less plausible. Despite the fact that these methods may miss some robust promising regions, the reported results of solving several $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$ problems using these methods in [28, 30, 68] indicate their superiority to methods that work based on the predicted fitness values of candidate solutions. However, there are still further concerns about these methods:

- The performance of these methods highly depends on how well the multiple moving promising regions can be tracked. Therefore, in cases where the tracking performance drops, the performance of these methods will deteriorate as well. For example, in problems with a large number of promising regions, performing efficient tracking is very challenging or even impossible.
- In some real-world problems, a considerable amount of data is available that can be used for predicting some other aspects of the future environments (e.g., data that can be used for predicting the next positions of optima). However, these ROOT methods only work on the basis of some specific gathered knowledge about the covered promising regions, they cannot benefit from the available data in such DOPs in their current form.

6) Multi-objective methods developed for $\mathrm{ROOT}^{\mathrm{S}}$ *problems:* Multi-objective methods, ROOT/SC [73] and ROOT/SCII [74] in particular, can be used for tackling both $\mathrm{ROOT}^{\mathrm{S}}$ problems that are multi-objective problems with one decision-maker and the ones that are single-objective by nature but are coupled with additional objectives in order to customize the search process based on some preferences. These methods search for POS in each environment and choose a solution from POS for deployment when the previous deployed solution is no longer acceptable. This strategy is more suitable for solving multi-objective $\mathrm{ROOT}^{\mathrm{S}}$ problems in which the preferences also change over time. Therefore, after knowing new preferences, a solution can be chosen from the POS. In [73, 74], the preferences are considered predefined and fixed. In such a circumstance, searching for POS may not be necessary. That is because the problem can be transformed into a single-objective problem by aggregating objectives based on the given preferences [83, 84]. Based on this alternative strategy, the ROOT method can focus on finding a solution for deployment in the resulting single-objective search space. Currently, there is no study that has investigated the strengths and weaknesses of these two strategies.

IV. Benchmarking ROOT methods

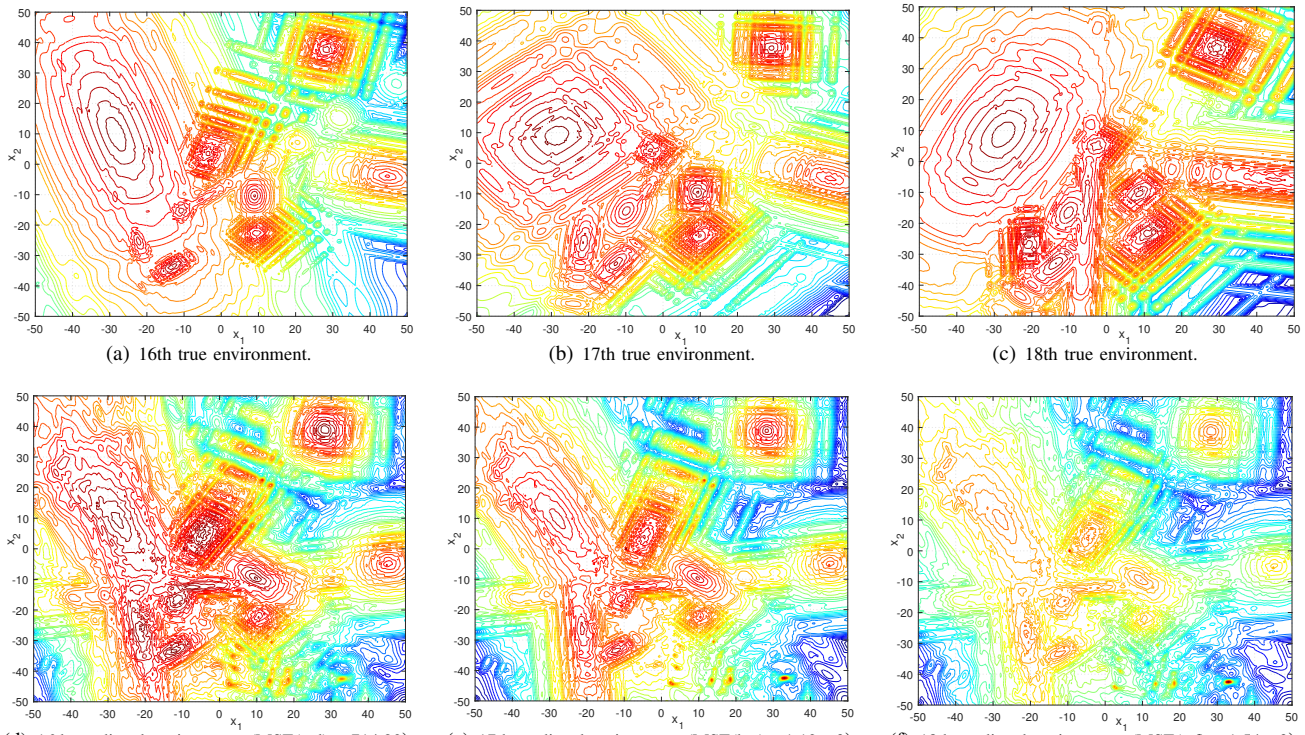
To evaluate the efficiency of ROOT methods, a variety of configurable benchmark generators and performance indicators have been used in the field. Configurable benchmark test suites are particularly useful in generating problem instances with controllable dynamical and morphological characteristics, making them essential for evaluating the performance of ROOT methods and investigating their strengths and weaknesses under various conditions. Besides, measuring the performance of ROOT methods is crucial to their development [85]. Performance indicators are used to evaluate the efficiency of ROOT methods, analyze their behavior in solving problems with varying characteristics, and compare their performance with existing methods. We provide a comprehensive review of the ROOT benchmarks and performance indicators used in the field in Sections S-I and S-II of the supplementary document, respectively.

V. Future research directions

By critically reviewing the current state of ROOT literature, we identify the following potential future research directions, which can broaden the field and dwindle some gaps between academic research and practice.

A. ROOT problems to investigate

1) ROOT^{S}_{Q}*/*ROOT^{S}_{G} *with changing quality threshold:* In real-world $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$/$\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{G}}$ problems, the quality threshold may change over time due to factors such as cost or demand fluctuations. Changing the quality threshold affects the algorithms' search process as they work based on this value (e.g., see (10) and (12)). Consequently, to tackle such problems, an algorithm needs to find several solutions that are suitable for various ranges of estimated future quality threshold values, which is computationally expensive. Tackling



(a) 16th true environment. (b) 17th true environment. (c) 18th true environment.

(d) 16th predicted environment (MSE(a,d) = 714.90). (e) 17th predicted environment (MSE(b,e) = 1.10e+3). (f) 18th predicted environment (MSE(c,f) = 1.54e+3).

Fig. 5. (a)–(c) show three successive 2-dimensional environments generated by generalized moving peaks benchmark (GMPB) [66, 67] after 15th, 16th, and 17th environmental changes, respectively. Note that the changes are generated by random-based dynamics [47]. To generate each figure, we used 40,000 points. True environments 1–15 are used to train predictors to estimate the future fitness values of the 40,000 points in environments 16–18, which are shown in (d)–(f), respectively. More specifically, for training the predictors that generate (d)–(f), we used the true fitness values in the 1–15th environments, the true fitness values in the 2–15th environments as well as the predicted fitness values in the 16th environment, and the true fitness values in the 3–15th environments plus the predicted fitness values in the 16–17th environments, respectively. We use the autoregressive integrated moving average (ARIMA) model [82] as the predictor. For each predicted plot, we also report the mean squared error (MSE) to the related true environment.

$\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$/$\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{G}}$ problems with changing quality thresholds is a challenging future research direction.

2) $ROOT^S_T$ with changing time window: In $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{T}}$, a deployed solution must be kept until the end of the time window. This temporal constraint is a result of some limitations, such as the lack of necessary resources (e.g., financial and/or human resources) for deploying a new solution until the end of the time window. Since the available and/or required resources for deploying a new solution may change over time, the length of the time window can also change and be revealed at the beginning of each time window. This will be challenging since the best solutions for deployment can be different for various sizes of time window. Thus, different solutions must be located where each of them is best for a specific estimated future time window size. Tackling $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{T}}$ problems with a changing time window size over time is a potential area for future research.

3) Multimodal $ROOT^M$: A multimodal ROOT problem with multiple decision-makers with various preferences can be categorized as a $\mathrm{ROOT}^{\mathrm{M}}$ problem. In these problems, there are multiple regions in the search space that contain global or near-global optima. These regions can be considered promising because they likely contain robust solutions that are candidates for deployment. However, since the regions containing each global optimum position can have different morphological and dynamical characteristics, it is important to evaluate the robustness and reliability of these regions to be used in determining the set of deployed solutions. Evaluating the robustness of these regions can help ensure that the deployed solutions are stable and reliable. Solving these problems is an interesting future direction.

4) ROOT in constrained DOPs: Despite the importance of constrained ROOT problems, little attention has been paid to them in the ROOT literature. In fact, most real-world applications are subject to some constraints. These problems are challenging as their objective functions and constraints usually change over time [31]. They can become further challenging when there are several disjointed moving feasible regions [86].

In constrained $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$/$\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{G}}$, in addition to the quality threshold that can be considered a constraint, there are also other constraints, which usually change over time. Consequently, the ROOT method not only needs to find feasible solutions in the current environment, but it also needs to take the feasibility of solutions in forthcoming environments into account. Solving constrained $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{T}}$ problems can be more challenging since the deployed solution must remain feasible for the entire duration of the time window because it cannot be changed during the time slot. Furthermore, constrained $\mathrm{ROOT}^{\mathrm{M}}$ can be very challenging, as we also need to consider the future feasibilities of the deployed set of solutions. In fact, the quality of the deployed set of solutions can drop quickly

if it loses many of its solutions due to becoming infeasible in the new environments. Hence, an important and challenging future direction is to investigate different types of constrained ROOT problems.

5) $ROOT^S$ problems with a large number of promising regions (peaks): The performance of all existing $\mathrm{ROOT^S}$ methods has only been tested on problems with a relatively small number of promising regions. Note that although the number of promising regions in benchmarks, such as mMPB, has been set to values up to 200, the number of visible promising regions in the landscape is considerably lower than this number. The reason is that the shape of promising regions is conical, and many smaller promising regions are covered by the larger ones.

In problem instances with a large number of promising regions, the performance of the methods that work based on estimating the robustness of promising regions will deteriorate because they cannot cover all promising regions [87]. Furthermore, as the problem becomes highly multimodal, the efficiency of methods that work based on estimating the future fitness/robustness of candidate solutions also drops due to the increased prediction error. In addition, performing exploration will be challenging in such problems due to the existence of a large number of promising regions. Working on $\mathrm{ROOT^S}$ problems with a large number of the promising regions will be an interesting future work.

6) Time-linkage ROOT: In problems with time-linkage property, the deployed solution in one environment influences the upcoming environments [88]. Among ROOT problems, $\mathrm{ROOT_G}$ problems, whose switching cost is defined based on the deployed and candidate solutions, possess the time-linkage feature [28]. However, in all existing works in the ROOT literature that consider the switching cost, the time-linkage property is not taken into account, which deteriorates the total gain over time [46]. Besides the time-linkage property caused by the switching cost, many real-world applications also have the time-linkage property, such as dynamic covering location problems [89, 90] in which the current positions of facilities affect the current and future search spaces and optimal solutions. Taking the time-linkage property into account in ROOT problems that possess this feature in order to improve the overall performance is a potential future direction.

7) ROOT in partially separable problems: Most existing works are focused on fully non-separable problems. However, the variable structure in many real-world applications is partially separable [91, 92]. It is shown in [77, 93] that the number of promising regions grows exponentially in partially separable objective functions. This results in an increase in the future fitness prediction error of the methods that work based on the estimated robustness of candidate solutions due to the roughness of the search space. These problems are also challenging for methods that work based on the estimated robustness of promising regions because tracking multiple moving promising regions will be less effective. To tackle partially separable ROOT problems, in the first step, the variable structure of the problem should be uncovered using a variable grouping method such as DG2 [94]. Thereafter, an EDOA must be assigned to each sub-function. To choose a solution for deployment, the ROOT algorithm must consider several factors for each sub-function, such as spatial severity, temporal severity, the contribution to the overall fitness value, and the optimization progress. Investigating partially separable ROOT problems is a potential future work.

8) ROOT in heterogeneous DOPs: Existing research in the literature has focused primarily on addressing problems with regular characteristics, such as fixed change frequencies and severities, and identical dynamics during environmental changes. However, in the real world, many DOPs exhibit irregular dynamics over time, with changes in severity, timing, and patterns over time. These DOPs are referred to as heterogeneous [10], and they present a significant challenge for finding robust solutions. In heterogeneous DOPs, the dynamical behavior of the problem changes over time, and the landscape can be subject to multiple types of dynamics. This means that deployed solutions need to be robust against various types of environmental changes. Furthermore, predicting the robustness of solutions in these problems is challenging due to the irregularity of their dynamics. The development of methods for solving heterogeneous DOPs represents an important area for future work.

9) ROOT problems with predictable characteristics: The majority of the existing works focus on problems with random-based dynamics for generating environmental changes. However, there are real-world problems with predictable characteristics, such as the reappearing ones [9, 72, 95], or the ones whose promising regions' movements are fully correlated, e.g., linear movements [80]. Indeed, if a ROOT method can benefit from the predictability properties of such problems, it may have some advantages over methods that do not take such properties into account. Investigating ROOT problems with predictable characteristics is a potential future direction.

10) Multi-objective $ROOT^S_Q$ problems with changing preferences: In any online multi-objective $\mathrm{ROOT^S_Q}$ problem with one decision-maker, regardless of the number of objective functions, only one solution is chosen for deployment when a new solution needs to be deployed. In solving multi-objective $\mathrm{ROOT^S_Q}$ with *stationary* preferences, finding the POS may not be necessary since the multi-objective problem can be transformed into a single-objective one based on the preferences. However, in multi-objective problems whose preferences also change over time, searching for the POS is necessary. In such problems, the deployed solution can also become unacceptable due to changes in preferences. When a solution is deployed, the algorithm continues exploitation and exploration for the next solution. Since the preferences are also changing over time, the algorithm needs to find the POS or a part of it according to the predicted range of the next preferences. Once the new preferences are known, the algorithm can focus on exploitation around the best solution in POS according to the current preferences. A future work will be to study multi-objective $\mathrm{ROOT^S_Q}$ problems with changing preferences.

11) $ROOT^M_T$ and $ROOT^M_G$: As stated in Section II, only $\mathrm{ROOT^M_Q}$ problems have been studied so far among $\mathrm{ROOT^M}$ problems, while $\mathrm{ROOT^M_T}$ and $\mathrm{ROOT^M_G}$ problems are also important because they can be found in the real world. $\mathrm{ROOT^M_T}$ represents real-world problems in which multiple

solutions are deployed simultaneously at the beginning of each time window and must remain operational until the end of the time window because changing the deployed set of solutions before that time is not possible. $\mathrm{ROOT}^{\mathrm{M}}_{\mathrm{G}}$, on the other hand, represents problems in which the costs and gains of switching the deployed set of solutions are considered. In such problems, the deployed set of solutions is replaced with a new set if either the quality of the previously deployed set is unacceptable or the benefit of switching outweighs the cost. Two important future research directions are designing ROOT methods for solving $\mathrm{ROOT}^{\mathrm{M}}_{\mathrm{T}}$ and $\mathrm{ROOT}^{\mathrm{M}}_{\mathrm{G}}$ problems.

B. Gaps in ROOT methods

1) Designing specified EDOAs for ROOT: Almost all existing ROOT methods use generic EDOAs, which were originally designed for tracking the moving optimum. Although the role of EDOAs in the ROOT methods is crucial, little attention has been given to designing components of EDOAs that take ROOT's considerations into account. For example, new EDOAs should be developed that can prioritize the areas containing robust solutions. Designing specialized EDOAs to be embedded in ROOT frameworks is an important future direction.

2) Quick recovery: Most existing ROOT methods choose a solution(s) for deployment at the *end of the environment* when a new solution(s) needs to be deployed, which is impractical [8, 32]. After an environmental change, if the deployed solution(s) is no longer acceptable in $\mathrm{ROOT}_{\mathrm{Q}}$ and $\mathrm{ROOT}_{\mathrm{G}}$, or a new time window has been started in $\mathrm{ROOT}_{\mathrm{T}}$, a new solution(s) must be chosen for deployment before a *deadline* (i.e., a temporal threshold). This requirement, known as *quick recovery*, can be found in many real-world problems [8, 96, 97]. To consider the quick recovery feature, an algorithm should focus on the areas containing robust solutions to improve the chosen solution for deployment before the deadline. Designing components for ROOT algorithms that consider quick recovery is a future direction.

3) Using both predicted fitness values of candidate solutions and estimated robustness of the promising regions: Hybridizing the ROOT methods that work based on the predicted future fitness of candidate solutions and estimated robustness of the promising regions in order to benefit from the strengths of both methods is another interesting future research direction. To this end, each sub-population that covers a promising region can have its own archive and local approximation component. The algorithm can use the predicted fitness values to increase the accuracy of determining the robustness of promising regions. Besides, the algorithm can also exploit the identified robust promising regions considering both current and predicted fitness values.

4) Multi-tasking: Multi-task optimization [98, 99] studies how to solve multiple optimization problems concurrently. These optimization problems, denoted as "tasks," are assumed to be related in some way, allowing for cross-task knowledge transfer that can improve the performance of solving each task independently [100]. In the context of ROOT problems, if there are multiple ROOT problems with a degree of relatedness, there are various information to gather and knowledge to acquire in each problem that can be used for improving the performance of finding robust solutions in other problems. By designing multi-task ROOT methods, it may be possible to leverage this knowledge to improve the performance of finding robust solutions in multiple ROOT problems simultaneously. It is important to note that while a multi-task ROOT solver may find multiple solutions for deployment simultaneously, each solution belongs to a separate ROOT problem. Therefore, the multi-task ROOT method does not change the class of those independent problems. Designing multi-task ROOT methods to solve multiple related ROOT problems is an interesting direction for future work.

C. Works to be done in benchmarking and assessment of ROOT methods

1) New performance indicator(s) for $\mathrm{ROOT}_{\mathrm{Q}}$*:* As stated in Section S-II-E of the supplementary document, the average survival time, which is the most well-known and commonly used performance indicator for $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$ and $\mathrm{ROOT}^{\mathrm{M}}_{\mathrm{Q}}$, suffers from several flaws. Consequently, an important future work will be to design new performance indicators for $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$ and $\mathrm{ROOT}^{\mathrm{M}}_{\mathrm{Q}}$. In designing new performance indicators, different factors should be taken into account, such as:

- The ratio between the number of deployed solutions/POSs and the total number of environments, and/or
- A penalty value for changing the deployed solutions/POSs in successive environments.

2) Designing new ROOT benchmark problems: Benchmark test suites play an important role in the development of dynamic optimization algorithms, including ROOT methods. As stated before, all continuous-space benchmarks used in the field are artificial. The field also lacks ROOT-specific benchmarks with discrete search space [101]. Designing ROOT-specific benchmark test suites that simulate or model real-world problems will provide a paradigm shift away from the current focus on artificial problems to solve real-world applications.

D. Challenges and progress in solving real-world ROOT problems

The field of ROOT aims to solve real-world problems, but it is still in its early stages. While progress has been made in developing algorithms, much work remains to be done before the field can fully address the challenges of real-world problems. Currently, the performance of algorithms is typically evaluated using artificial benchmark problems and some existing ROOT methods have been applied to real-world-based benchmark problems, including nonlinear dynamic stochastic optimization problems for stochastic energy management [102, 103] and dynamic customer location-allocation [104, 105]. One of the main challenges in solving real-world problems is that dynamic handling components are not yet capable of handling the complexities of these problems, such as irregular changes over time, multiple types of environmental changes, local and hard-to-detect environmental changes, and continuously changing environments. Addressing these challenges will require the

development of more complex algorithms. While the field works toward developing more complex algorithms, simpler scenarios of important real-world optimization problems, such as dynamic covering location problems [44], can be tackled as future work. Solving these problems will provide valuable insight into how ROOT can be used to tackle more complex real-world problems in the future.

VI. Conclusion

In this survey, we provided an in-depth review of robust optimization over time (ROOT) [11]. We aimed to help researchers and practitioners gain a broad perspective on the current state of the field, what has been achieved since ROOT was first introduced in 2010 [11], and the shortcomings and issues of the field. In this survey, we proposed two taxonomies for ROOT problems and methods, respectively. We then comprehensively reviewed these problems and methods according to the proposed taxonomies. We also reviewed commonly used benchmark problems and performance indicators. By gaining insights from reviewing the current state of ROOT literature, we finally highlighted some important potential research directions that can broaden the field.

References

[1] H.-G. Beyer and B. Sendhoff, "Robust optimization–a comprehensive survey," *Computer methods in applied mechanics and engineering*, vol. 196, no. 33-34, pp. 3190–3218, 2007.

[2] J. E. Ward and R. E. Wendell, "Approaches to sensitivity analysis in linear programming," *Annals of Operations Research*, vol. 27, no. 1, pp. 3–38, 1990.

[3] H. Xu and S. Mannor, "Robustness and generalization," *Machine learning*, vol. 86, no. 3, pp. 391–423, 2012.

[4] M. Kalsi, K. Hacker, and K. Lewis, "A comprehensive robust design approach for decision trade-offs in complex systems design," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 19715. American Society of Mechanical Engineers, 1999, pp. 1343–1354.

[5] J. K. Allen, C. Seepersad, H. Choi, and F. Mistree, "Robust Design for Multiscale and Multidisciplinary Applications," *Journal of Mechanical Design*, vol. 128, no. 4, pp. 832–843, 01 2006.

[6] Y. Jin, K. Tang, X. Yu, B. Sendhoff, and X. Yao, "A framework for finding robust optimal solutions over time," *Memetic Computing*, vol. 5, no. 1, pp. 3–18, 2013.

[7] J. C. Helton, "Uncertainty and sensitivity analysis in the presence of stochastic and subjective uncertainty," *Journal of Statistical Computation and Simulation*, vol. 57, no. 1-4, pp. 3–76, 1997.

[8] T. T. Nguyen, "Continuous dynamic optimisation using evolutionary algorithms," Ph.D. dissertation, University of Birmingham, 2011.

[9] J. Branke, *Evolutionary optimization in dynamic environments*. Springer Science & Business Media, 2012, vol. 3.

[10] D. Yazdani, R. Cheng, D. Yazdani, J. Branke, Y. Jin, and X. Yao, "A survey of evolutionary continuous dynamic optimization over two decades – part A," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 609–629, 2021.

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

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

[13] ——, "Finding robust solutions to dynamic optimization problems," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2013, pp. 616–625.

[14] D. Yazdani, J. Branke, M. N. Omidvar, T. T. Nguyen, and X. Yao, "Changing or keeping solutions in dynamic optimization problems with switching costs," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2018, pp. 1095–1102.

[15] M. Chen, Y. Guo, H. Liu, and C. Wang, "The evolutionary algorithm to find robust pareto-optimal solutions over time," *Mathematical Problems in Engineering*, vol. 2015, 2015.

[16] Y. Guo, H. Yang, M. Chen, J. Cheng, and D. Gong, "Ensemble prediction-based dynamic robust multi-objective optimization methods," *Swarm and Evolutionary Computation*, vol. 48, pp. 156–171, 2019.

[17] Y. Guo, H. Yang, M. Chen, D. Gong, and S. Cheng, "Grid-based dynamic robust multi-objective brain storm optimization algorithm," *Soft Computing*, vol. 24, no. 10, pp. 7395–7415, 2020.

[18] M. Chen, Y. Guo, Y. Jin, S. Yang, D. Gong, and Z. Yu, "An environment-driven hybrid evolutionary algorithm for dynamic multi-objective optimization problems," *Complex & Intelligent Systems*, pp. 1–17, 2022.

[19] X. Li, M. G. Epitropakis, K. Deb, and A. Engelbrecht, "Seeking multiple solutions: An updated survey on niching methods and their applications," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 4, pp. 518–538, 2016.

[20] X. Lin, W. Luo, P. Xu, Y. Qiao, and S. Yang, "Popdmmo: A general framework of population-based stochastic search algorithms for dynamic multimodal optimization," *Swarm and Evolutionary Computation*, vol. 68, p. 101011, 2022.

[21] A. Ahrari, S. Elsayed, R. Sarker, D. Essam, and C. A. C. Coello, "Static and dynamic multimodal optimization by improved covariance matrix self-adaptation evolution strategy with repelling subpopulations," *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 3, pp. 527–541, 2021.

[22] C. Cruz, J. R. González, and D. A. Pelta, "Optimization in dynamic environments: a survey on problems, methods and measures," *Soft Computing*, vol. 15, no. 7, pp. 1427–1448, 2011.

[23] M. Mavrovouniotis, C. Li, and S. Yang, "A survey of swarm intelligence for dynamic optimization: Algorithms and applications," *Swarm and Evolutionary Computation*, vol. 33, pp. 1 – 17, 2017.

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

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

[26] L. Zhu, K. Deb, and S. Kulkarni, "Multi-scenario optimization using multi-criterion methods: A case study on byzantine agreement problem," in *IEEE Congress on Evolutionary Computation*. IEEE, 2014, pp. 2601–2608.

[27] B. Shavazipour, J. H. Kwakkel, and K. Miettinen, "Multi-scenario multi-objective robust optimization under deep uncertainty: A posteriori approach," *Environmental Modelling & Software*, vol. 144, p. 105134, 2021.

[28] D. Yazdani, "Particle swarm optimization for dynamically changing environments with particular focus on scalability and switching cost," Ph.D. dissertation, Liverpool John Moores University, Liverpool, UK, 2018.

[29] Z.-H. Zhan, L. Shi, K. C. Tan, and J. Zhang, "A survey on evolutionary computation for complex continuous optimization," *Artificial Intelligence Review*, pp. 1–52, 2022.

[30] D. Yazdani, T. T. Nguyen, and J. Branke, "Robust optimization over time by learning problem space characteristics," *IEEE*

Transactions on Evolutionary Computation, vol. 23, no. 1, pp. 143–155, 2019.

[31] T. T. Nguyen and X. Yao, "Continuous dynamic constrained optimization—the challenges," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 6, pp. 769–786, 2012.

[32] D. Yazdani, D. Yazdani, J. Branke, M. N. Omidvar, Amir H. Gandomi, and X. Yao, "Robust optimization over time by estimating robustness of promising regions," *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 3, pp. 657–670, 2022.

[33] K. De Jong, "Evolving in a changing world," in *Foundations of Intelligent Systems: 11th International Symposium, ISMIS'99 Warsaw, Poland, June 8–11, 1999 Proceedings 11*. Springer, 1999, pp. 512–519.

[34] C. D. Laird and L. T. Biegler, "Large-scale nonlinear programming for multi-scenario optimization," in *Modeling, Simulation and Optimization of Complex Processes: Proceedings of the Third International Conference on High Performance Scientific Computing, March 6–10, 2006, Hanoi, Vietnam*. Springer, 2008, pp. 323–336.

[35] H. Wang, J. Doherty, and Y. Jin, "Hierarchical surrogate-assisted evolutionary multi-scenario airfoil shape optimization," in *IEEE Congress on Evolutionary Computation*. IEEE, 2018, pp. 1–8.

[36] K. Deb, L. Zhu, and S. Kulkarni, "Multi-scenario, multi-objective optimization using evolutionary algorithms: Initial results," in *IEEE Congress on Evolutionary Computation*. IEEE, 2015, pp. 1877–1884.

[37] W. Li, R. Wang, T. Zhang, M. Ming, and H. Lei, "Multi-scenario microgrid optimization using an evolutionary multi-objective algorithm," *Swarm and Evolutionary Computation*, vol. 50, p. 100570, 2019.

[38] X. Zhou, H. Wang, W. Peng, B. Ding, and R. Wang, "Solving multi-scenario cardinality constrained optimization problems via multi-objective evolutionary algorithms," *Science China Information Sciences*, vol. 62, pp. 1–18, 2019.

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

[40] Y. Jin, H. Wang, and C. Sun, *Data-Driven Evolutionary Optimization*. Springer, 2021.

[41] S. Chatterjee and A. S. Hadi, *Regression analysis by example*. John Wiley & Sons, 2006.

[42] H. Fu, "Finding robust solutions against environmental changes," Ph.D. dissertation, University of Birmingham, 2014.

[43] C. Martella, J. Li, C. Conrado, and A. Vermeeren, "On current crowd management practices and the need for increased situation awareness, prediction, and intervention," *Safety Science*, vol. 91, pp. 381 – 393, 2017.

[44] F. Plastria, *Covering Location Problems*. Springer-Verlag New York, 2002, pp. 37–79.

[45] M. Haghani, E. Kuligowski, A. Rajabifard, and P. Lentini, "Fifty years of scholarly research on terrorism: Intellectual progression, structural composition, trends and knowledge gaps of the field," *International Journal of Disaster Risk Reduction*, vol. 68, p. 102714, 2022.

[46] D. Yazdani, T. T. Nguyen, J. Branke, and J. Wang, "A multi-objective time-linkage approach for dynamic optimization problems with previous-solution displacement restriction," in *Applications of Evolutionary Computation*, K. Sim and P. Kaufmann, Eds. Springer International Publishing, 2018, pp. 864–878.

[47] D. Yazdani, R. Cheng, D. Yazdani, J. Branke, Y. Jin, and X. Yao, "A survey of evolutionary continuous dynamic optimization over two decades – part B," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 630–650, 2021.

[48] M. N. Omidvar, X. Li, and X. Yao, "A review of population-based metaheuristics for large-scale black-box global optimization—Part I," *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 5, pp. 802–822, 2021.

[49] ——, "A review of population-based metaheuristics for large-scale black-box global optimization—Part II," *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 5, pp. 823–843, 2021.

[50] Y. Jin, H. Wang, T. Chugh, D. Guo, and K. Miettinen, "Data-driven evolutionary optimization: An overview and case studies," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 3, pp. 442–458, 2018.

[51] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft computing*, vol. 9, no. 1, pp. 3–12, 2005.

[52] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-International Conference on Neural Networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[53] R. C. Eberhart and Y. Shi, "Tracking and optimizing dynamic systems with particle swarms," in *IEEE Congress on Evolutionary Computation*, vol. 1. IEEE, 2001, pp. 94–100.

[54] M. Fox, S. Yang, and F. Caraffini, "An experimental study of prediction methods in robust optimization over time," in *IEEE Congress on Evolutionary Computation*. IEEE Press, 2020.

[55] J.-Y. Guzmán-Gaspar and E. Mezura-Montes, "Robust optimization over time with differential evolution using an average time approach," in *IEEE Congress on Evolutionary Computation*. IEEE, 2019, pp. 1548–1555.

[56] ——, "Differential evolution variants in robust optimization over time," in *2019 International Conference on Electronics, Communications and Computers (CONIELECOMP)*. IEEE, 2019, pp. 164–169.

[57] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2010.

[58] P. Novoa-Hernández, D. A. Pelta, and C. C. Corona, "Approximation models in robust optimization over time-an experimental study," in *IEEE Congress on Evolutionary Computation*. IEEE, 2018, pp. 1–6.

[59] K. Deb and H. Gupta, "Introducing robustness in multi-objective optimization," *Evolutionary Computation*, vol. 14, no. 4, pp. 463–494, 2006.

[60] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

[61] I. O. Essiet and Y. Sun, "Tracking variable fitness landscape in dynamic multi-objective optimization using adaptive mutation and crossover operators," *IEEE Access*, vol. 8, pp. 188 927–188 937, 2020.

[62] Y. Shi, "Brain storm optimization algorithm," in *Advances in Swarm Intelligence*, Y. Tan et al., Ed. Springer Berlin Heidelberg, 2011, pp. 303–309.

[63] W. Mkaouer, M. Kessentini, A. Shaout, P. Koligheu, S. Bechikh, K. Deb, and A. Ouni, "Many-objective software remodularization using nsga-iii," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 24, no. 3, pp. 1–45, 2015.

[64] J.-Y. Guzmán-Gaspar, E. Mezura-Montes, and S. Domínguez-Isidro, "Differential evolution in robust optimization over time using a survival time approach," *Mathematical and Computational Applications*, vol. 25, no. 4, p. 72, 2020.

[65] R. Poli, "Mean and variance of the sampling distribution of particle swarm optimizers during stagnation," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 4, pp. 712–721, 2009.

[66] D. Yazdani, M. N. Omidvar, R. Cheng, J. Branke, T. T. Nguyen, and X. Yao, "Benchmarking continuous dynamic optimization: Survey and generalized test suite," *IEEE Transactions on Cybernetics*, vol. 52, no. 5, pp. 3380–3393, 2022.

[67] D. Yazdani, J. Branke, M. N. Omidvar, X. Li, C. Li, M. Mavrovouniotis, T. T. Nguyen, S. Yang, and X. Yao, "IEEE

CEC 2022 competition on dynamic optimization problems generated by generalized moving peaks benchmark," *arXiv preprint arXiv:2106.06174*, 2021.

[68] D. Yazdani, T. T. Nguyen, J. Branke, and J. Wang, "A new multi-swarm particle swarm optimization for robust optimization over time," in *Applications of Evolutionary Computation*, G. Squillero and K. Sim, Eds. Springer International Publishing, 2017, pp. 99–109.

[69] L. Adam and X. Yao, "A simple yet effective approach to robust optimization over time," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2019, pp. 680–688.

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

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

[72] B. Nasiri, M. Meybodi, and M. Ebadzadeh, "History-driven particle swarm optimization in dynamic and uncertain environments," *Neurocomputing*, vol. 172, pp. 356 – 370, 2016.

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

[74] Y. Huang, Y. Jin, and K. Hao, "Decision-making and multi-objectivization for cost sensitive robust optimization over time," *Knowledge-Based Systems*, p. 105857, 2020.

[75] X. Hu and R. C. Eberhart, "Adaptive particle swarm optimization: detection and response to dynamic systems," in *IEEE Congress on Evolutionary Computation*, vol. 2. IEEE, 2002, pp. 1666–1670.

[76] D. Yazdani, B. Nasiri, A. Sepas-Moghaddam, M. Meybodi, and M. Akbarzadeh-Totonchi, "mNAFSA: a novel approach for optimization in dynamic environments with global changes," *Swarm and Evolutionary Computation*, vol. 18, pp. 38 – 53, 2014.

[77] D. Yazdani, M. N. Omidvar, J. Branke, T. T. Nguyen, and X. Yao, "Scaling up dynamic optimization problems: A divide-and-conquer approach," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 1, pp. 1–15, 2020.

[78] D. Yazdani, R. Cheng, C. He, and J. Branke, "Adaptive control of subpopulations in evolutionary dynamic optimization," *IEEE Transactions on Cybernetics*, vol. 52, no. 7, pp. 6476–6489, 2022.

[79] Q. Y. Kenny, W. Li, and A. Sudjianto, "Algorithmic construction of optimal symmetric latin hypercube designs," *Journal of Statistical Planning and Inference*, vol. 90, no. 1, pp. 145–159, 2000.

[80] P. Angeline, "Tracking extrema in dynamic environments," in *Evolutionary Programming VI*, P. Angeline et al., Ed. Springer Lecture Notes in Computer Science, 1997, vol. 1213, pp. 335–345.

[81] C. Li and S. Yang, "A generalized approach to construct benchmark problems for dynamic optimization," in *Simulated Evolution and Learning*, X. L. et al., Ed. Springer Lecture Notes in Computer Science, 2013, vol. 5361, pp. 391–400.

[82] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[83] R. T. Marler and J. S. Arora, "The weighted sum method for multi-objective optimization: new insights," *Structural and multidisciplinary optimization*, vol. 41, no. 6, pp. 853–862, 2010.

[84] S. Kaddani, D. Vanderpooten, J.-M. Vanpeperstraete, and H. Aissi, "Weighted sum model with partial preference information: application to multi-objective optimization," *European Journal of Operational Research*, vol. 260, no. 2, pp. 665–679, 2017.

[85] Y. Huang, Y. Jin, and Y. Ding, "New performance indicators for robust optimization over time," in *IEEE Congress on Evolutionary Computation*. IEEE, 2015, pp. 1380–1387.

[86] C. Bu, W. Luo, and L. Yue, "Continuous dynamic constrained optimization with ensemble of locating and tracking feasible regions strategies," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 1, pp. 14–33, 2016.

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

[88] W. Zheng, H. Chen, and X. Yao, "Analysis of evolutionary algorithms on fitness function with time-linkage property," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 696–709, 2021.

[89] J. Brimberg, P. Hansen, N. Mladenovic, and S. Salhi, "A survey of solution methods for the continuous location-allocation problem," *International Journal of Operations Research*, vol. 5, no. 1, pp. 1 – 12, 2008.

[90] T. T. Nguyen and X. Yao, "Dynamic time-linkage evolutionary optimization: Definitions and potential solutions," in *Metaheuristics for Dynamic Optimization*. Springer, 2013, pp. 371–395.

[91] M. N. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative co-evolution with differential grouping for large scale optimization," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 378–393, 2014.

[92] M. N. Omidvar, X. Li, and K. Tang, "Designing benchmark problems for large-scale continuous optimization," *Information Sciences*, vol. 316, pp. 419–436, 2015.

[93] M. N. Omidvar, D. Yazdani, J. Branke, X. Li, S. Yang, and X. Yao, "Generating large-scale dynamic optimization problem instances using the generalized moving peaks benchmark," *arXiv preprint arXiv:2107.11019*, 2021.

[94] M. N. Omidvar, M. Yang, Y. Mei, X. Li, and X. Yao, "DG2: A faster and more accurate differential grouping for large-scale black-box optimization," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 6, pp. 929–942, 2017.

[95] T. Zhu, W. Luo, and L. Yue, "Combining multipopulation evolutionary algorithms with memory for dynamic optimization problems," in *IEEE Congress on Evolutionary Computation*. IEEE, 2014, pp. 2047–2054.

[96] L. Liu, S. R. Ranjithan, and G. Mahinthakumar, "Contamination source identification in water distribution systems using an adaptive dynamic optimization procedure," *Journal of Water Resources Planning and Management*, vol. 137, no. 2, pp. 183–192, 2011.

[97] L. Liu, E. M. Zechman, E. D. Brill, Jr, G. Mahinthakumar, S. Ranjithan, and J. Uber, "Adaptive contamination source identification in water distribution systems using an evolutionary algorithm-based dynamic optimization procedure," in *Water Distribution Systems Analysis Symposium*, 2008, pp. 1–9.

[98] B. Da, Y.-S. Ong, L. Feng, A. K. Qin, A. Gupta, Z. Zhu, C.-K. Ting, K. Tang, and X. Yao, "Evolutionary multitasking for single-objective continuous optimization: Benchmark problems, performance metric, and baseline results," *arXiv preprint arXiv:1706.03470*, 2017.

[99] A. Gupta, Y.-S. Ong, and L. Feng, "Multifactorial evolution: toward evolutionary multitasking," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 3, pp. 343–357, 2015.

[100] X. Zheng, A. K. Qin, M. Gong, and D. Zhou, "Self-regulated evolutionary multitask optimization," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 1, pp. 16–28, 2019.

[101] M. Mavrovouniotis, "Ant colony optimization in stationary and dynamic environments," Ph.D. dissertation, University of Leicester, 2013.

[102] Y. Liu and H. Liang, "A ROOT approach for stochastic energy management in electric bus transit center with PV and ESS," in *IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.
[103] ——, "A three-layer stochastic energy management approach for electric bus transit centers with PV and energy storage systems," *IEEE Transactions on Smart Grid*, vol. 12, no. 2, pp. 1346–1357, 2020.
[104] R. Ankrah, B. Lacroix, J. McCall, A. Hardwick, and A. Conway, "Introducing the dynamic customer location-allocation problem," in *IEEE Congress on Evolutionary Computation*. IEEE, 2019, pp. 3157–3164.
[105] R. Ankrah, B. Lacroix, J. McCall, A. Hardwick, A. Conway, and G. Owusu, "Racing strategy for the dynamic-customer location-allocation problem," in *IEEE Congress on Evolutionary Computation*. IEEE, 2020, pp. 1–8.

Supplementary Document of 'Robust Optimization Over Time: A Critical Review'

Danial Yazdani, *Member, IEEE,* Mohammad Nabi Omidvar, *Senior member, IEEE,* Donya Yazdani, Jürgen Branke, Trung Thanh Nguyen, Amir H. Gandomi, *Senior member, IEEE,* Yaochu Jin, *Fellow, IEEE,* and Xin Yao, *Fellow, IEEE*

CONTENTS

S-I. ROOT BENCHMARK PROBLEMS

A typical benchmark problem for robust optimization over time (ROOT) consists of three main components: baseline functions, dynamics, and ROOT-specific elements, including the quality threshold and switching cost. The dynamics used in these benchmark problems are those that are commonly used in the field of tracking the moving optimum (TMO), and as such, they are not discussed in this survey. For a comprehensive list of the dynamics used in the field and their characteristics, the reader is referred to [1]. Below, we provide a review of the baseline functions used in ROOT benchmarks, as well as the methodologies used to model the quality threshold and switching cost.

Danial Yazdani is with the Faculty of Engineering & Information Technology, University of Technology Sydney, Ultimo 2007, Australia. (e-mail: danial.yazdani@gmail.com)

Mohammad Nabi Omidvar is with the School of Computing, University of Leeds, and Leeds University Business School, Leeds LS2 9JT, United Kingdom. (e-mail: m.n.omidvar@leeds.ac.uk)

Donya Yazdani is with the Department of Computer Science, University of Sheffield, Sheffield S1 4DP, United Kingdom. (e-mail: dyazdani1@sheffield.ac.uk)

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

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

Amir H. Gandomi is with the Faculty of Engineering & Information Technology, University of Technology Sydney, Ultimo 2007, Australia. He is also with the University Research and Innovation Center (EKIK), Obuda University, Budapest 1034, Hungary. (e-mail: Gandomi@uts.edu.au)

Yaochu Jin is with the Faculty of Technology, Bielefeld University, Bielefeld 33615, Germany. (e-mail: yaochu.jin@uni-bielefeld.de)

Xin Yao is with the Research Institute of Trustworthy Autonomous Systems (RITAS), and Guangdong Provincial Key Laboratory of Brain inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China. He is also with the CERCIA, School of Computer Science, Birmingham B15 2TT, United Kingdom. (e-mail: xiny@sustech.edu.cn)

A. Baseline functions used in ROOT benchmarks

Various benchmarks have been used in the field for evaluating the effectiveness of $\mathrm{ROOT}^{\mathrm{M}}$ and $\mathrm{ROOT}^{\mathrm{S}}$ methods. Prior studies that concentrate on solving $\mathrm{ROOT}^{\mathrm{M}}$ problems, including [2–5], have utilized several commonly used benchmark problems in the field of dynamic multi-objective optimization [6–8]. These multi-objective benchmarks were originally employed in the field of evolutionary dynamic multi-objective optimization for tracking Pareto-optimal solutions (POS) over time and have been employed in ROOT literature without modification. Consequently, in this survey, we do not delve into their details. Surveys, which cover dynamic multi-objective benchmarks, can be referred to [9, 10]. On the other hand, studies focusing on solving $\mathrm{ROOT}^{\mathrm{S}}$ problems used benchmark problems specifically designed or modified to generate $\mathrm{ROOT}^{\mathrm{S}}$ problem instances. In the following, we describe the $\mathrm{ROOT}^{\mathrm{S}}$ benchmarks' baseline functions used in the field.

The first dedicated benchmark for $\mathrm{ROOT}^{\mathrm{S}}$ in the literature is the modified Moving Peaks Benchmark (mMPB) [11–15]. The landscapes produced by mMPB are created by combining multiple promising regions (peaks). A $\max(\cdot)$ function is normally used to define the basin of attraction of each promising region. The promising regions in mMPB are regular, smooth, symmetric, unimodal, and conical, with dynamically changing properties such as height, width, and location. The baseline function of mMPB is defined as:

$$f^{(t)}(\mathbf{x}) = \max_{i\in\{1,\ldots,m\}} \left\{ h_i^{(t)} - w_i^{(t)} \left\| \mathbf{x} - \mathbf{c}_i^{(t)} \right\| \right\}, \qquad \text{(S-1)}$$

where m is the number of promising regions, $\mathbf{x}$ is a solution in the d-dimensional problem space, $h_i^{(t)}$, $w_i^{(t)}$, and $\mathbf{c}_i^{(t)}$ are the height, width, and the center/summit location of the ith promising region in the tth environment, respectively. Note that the baseline functions in mMPB and MPB are identical, but their dynamics are different. In mMPB, the height, width,

and center of the ith promising region change from one environment to the next using the following equations:

$$h_i^{(t+1)} = h_i^{(t)} + \tilde{h}_i \, \mathrm{N}(0,1), \tag{S-2}$$

$$w_i^{(t+1)} = w_i^{(t)} + \tilde{w}_i \, \mathrm{N}(0,1), \tag{S-3}$$

$$\mathbf{c}_i^{(t+1)} = \mathbf{c}_i^{(t)} + \mathbf{v}_i^{(t+1)}, \tag{S-4}$$

$$\mathbf{v}_i^{(t+1)} = \tilde{s}_i \cdot \frac{(1-\lambda)\cdot \mathbf{u} + \lambda \cdot \mathbf{v}_i^{(t)}}{\left\| (1-\lambda)\cdot \mathbf{u} + \lambda \cdot \mathbf{v}_i^{(t)} \right\|}, \tag{S-5}$$

where $\tilde{h}_i$, $\tilde{w}_i$, and $\tilde{s}_i$ are change severity parameters of height, width, and shift of the ith promising region, $\mathbf{u}$ is a vector of uniformly distributed numbers in $[-0.5, 0.5]$, $\mathrm{N}(0,1)$ is a random number drawn from a Gaussian distribution with mean 0 and variance 1, and $\lambda \in (0,1)$ is a correlation coefficient. As can be seen in (S-2)-(S-5), unlike the traditional MPB in which all promising regions share the same change severity, each promising region i in mMPB has its own dedicated severity values. These differences result in different levels of robustness among promising regions [12].

Currently, mMPB is the most commonly used benchmark problem in the $\mathrm{ROOT}^{\mathrm{S}}$ literature, and has been used for evaluating the performance of $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$ [16, 17], $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{T}}$ [18], and $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{G}}$ [19] methods. However, it has the following shortcomings:

1) Its landscape is composed of promising regions that are easy to optimize due to their regularity, unimodality, symmetry, lack of ill-conditioning, and full separability [20, 21], which is not the case in most real-world problems.
2) Equation (S-5) shows that after each environmental change, the promising region center $\mathbf{c}_i$ is shifted with a fixed Euclidean length $\tilde{s}_i$, and only the direction of movement can be random (if λ is set to smaller values). According to our investigations, this property, which cannot be found in most real-world problems, causes a bias in favor of some methods that work based on the estimated robustness of promising regions.
3) Although the optimal fitness value in the landscapes generated by mMPB is known in each environment[1], the optimal value of robustness in mMPB is not known.

To address the first shortcoming of mMPB, the baseline function of the generalized moving peaks benchmark (GMPB) [21, 22] is used in [23]. Unlike MPB, the promising regions generated by GMPB can be highly complex (see Figs. 4(a)). GMPB's baseline function is defined as:

$$f^{(t)}(\mathbf{x}) = \max_{k\in\{1,\dots,m\}} \left\{ h_k^{(t)} - \sqrt{\mathbb{T}\left(\left(\mathbf{x}-\mathbf{c}_k^{(t)}\right)^{\top} \mathbf{R}_k^{(t)\top}, k\right) \; \mathbf{W}_k^{(t)} \; \mathbb{T}\left(\mathbf{R}_k^{(t)}\left(\mathbf{x}-\mathbf{c}_k^{(t)}\right), k\right)} \right\}, \tag{S-6}$$

[1]The global optimum position is the center of the promising region with the maximum height.

where $\mathbb{T}(\mathbf{y}, k) : \mathbb{R}^d \mapsto \mathbb{R}^d$ is calculated as:

$$\mathbb{T}(y_j, k) = \begin{cases} \exp\left(\log(y_j) + \tau_k^{(t)}\left(\sin\left(\eta_{k,1}^{(t)}\log(y_j)\right) + \sin\left(\eta_{k,2}^{(t)}\log(y_j)\right)\right)\right) & \text{if } y_j > 0 \\ 0 & \text{if } y_j = 0 \\ -\exp\left(\log(|y_j|) + \tau_k^{(t)}\left(\sin\left(\eta_{k,3}^{(t)}\log(|y_j|)\right) + \sin\left(\eta_{k,4}^{(t)}\log(|y_j|)\right)\right)\right) & \text{if } y_j < 0 \end{cases} \tag{S-7}$$

where $\eta_{k,l\in\{1,2,3,4\}}^{(t)}$ and $\tau_k^{(t)}$ are irregularity parameters of the kth promising region, $\mathbf{R}_k^{(t)}$ is the rotation matrix of kth promising region, and $\mathbf{W}_k^{(t)}$ is a $d \times d$ diagonal matrix whose diagonal elements show the width of kth promising region in different dimensions. Similar to mMPB, (S-2) and (S-3) are used to change the height and width of the promising regions, respectively. However, instead of (S-4) and (S-5), the following formula is used for relocating the centers of promising regions:

$$c_{i,j}^{(t+1)} = c_{i,j}^{(t)} + \mathrm{N}\left(0, \frac{\tilde{s}_i}{\sqrt{d}}\right), \tag{S-8}$$

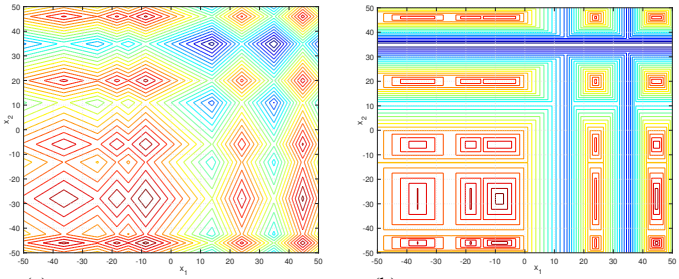
where $c_{i,j}^{(t)}$ is the jth dimension of the center position of the ith promising region. Using (S-8) instead of (S-4) and (S-5), results in shifting the center of promising regions with various lengths, which addresses the second shortcoming of mMPB mentioned above. Note that the third shortcoming of the mMPB has not been addressed in the modified GMPB used in [23] since the optimal robustness values are still unknown in this benchmark.

As knowing the optimal robustness values is desirable to measure the difference between the performance of an algorithm and the optimal performance, Fu et al. [24] proposed two benchmark problems in which the optimal robustness values are known. The baseline functions of these benchmarks, which are designed by modifying the baseline function of MPB, are as follows:

$$f^{(t)}(\mathbf{x}) = \frac{1}{d}\sum_{j=1}^{d}\left\{ \max_{i\in\{1,\dots,m\}} \left\{ h_{i,j}^{(t)} - w_{i,j}^{(t)} \left| x_j - c_{i,j}^{(t)} \right| \right\} \right\}, \tag{S-9}$$

$$f^{(t)}(\mathbf{x}) = \min_{j\in\{1,\dots,d\}} \left\{ \max_{i\in\{1,\dots,m\}} \left\{ h_{i,j}^{(t)} - w_{i,j}^{(t)} \left| x_j - c_{i,j}^{(t)} \right| \right\} \right\}, \tag{S-10}$$

where m is the number of promising regions along each dimension, d is the number of dimensions, and $h_{i,j}^{(t)}$, $w_{i,j}^{(t)}$, and $c_{i,j}^{(t)}$ are the height, width, and center of the ith promising region of the jth dimension in the tth environment, respectively. The baseline function in (S-9) is designed specifically for generating $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{T}}$ problem instances in which the optimal average fitness value by (7) can be mathematically calculated. Equation (S-10), on the other hand, is specifically designed for $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$ whose optimal average survival time in (10) can be mathematically calculated. To generate environmental changes, the dynamics in [25], including random, small step, large step, chaotic, circular, and noisy circular, are used in these two benchmarks. Although the possibility of calculating the optimal average fitness and survival time in (S-9) and (S-10) is desirable, these benchmark problems still suffer from the following shortcomings:



(a) A landscape generated by (S-9). (b) A landscape generated by (S-10).

Fig. S-1. 2-dimensional landscapes generated by (S-9) and (S-10) with the same parameter settings (for all promising regions' center positions, heights, and widths). m has been set to 5 which results in generating $5^2 = 25$ promising regions in a 2-dimensional space.

- The $L2$-norm distance in the standard MPB, which is the core of many real-world problems [26] such as continuous covering location problems [27], has been replaced by dimension-wise absolute values in (S-9) and (S-10), resulting in the generation of more artificial problems.
- The number of promising regions in the landscapes generated by these baseline functions is not controllable and can be up to m^d (some might be hidden beneath larger nearby promising regions [21]). Consequently, these baseline functions may not be suitable to generate problem instances with higher dimensions. This has resulted in using these benchmarks only to generate 2-dimensional problems in the literature [24, 28]. Figure S-1 depicts two examples of the landscapes generated by (S-9) and (S-10).

B. Definition of quality threshold and switching cost in ROOT benchmarks

1) Quality threshold definition in ROOT benchmarks: As stated in Section II, the quality threshold is a part of $\mathrm{ROOT}_{\mathrm{Q}}$ and $\mathrm{ROOT}_{\mathrm{G}}$ problems, which determines the acceptability of solutions, particularly the deployed solution. Various approaches have been used in designing ROOT benchmark problems to define the quality thresholds. In the existing $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{S}}$ benchmark problems, two different quality threshold definitions have been used in [13] and [18]. In [13], the acceptability of the deployed solution $\mathbf{s}$ in the tth environment is determined using:

$$\mathfrak{a}^{(t)}(\mathbf{s}) = \begin{cases} 0, & \text{if } \left| \dfrac{f^{(t)}\left(\mathbf{x}^{\star(t)}\right) - f^{(t)}(\mathbf{s})}{f^{(t)}\left(\mathbf{x}^{\star(t)}\right)} \right| > \delta \\ 1, & \text{otherwise} \end{cases}, \tag{S-11}$$

where $\mathfrak{a}^{(t)}(\mathbf{s}) = 1$ indicates that the deployed solution $\mathbf{s}$ is acceptable in the tth environment, and $\mathfrak{a}^{(t)}(\mathbf{s}) = 0$ otherwise, $\mathbf{x}^{\star(t)}$ is the optimum position in the tth environment, and δ denotes the threshold of the maximum tolerated fitness difference between $\mathbf{s}$ and $\mathbf{x}^{\star(t)}$. However, this definition for determining the acceptability of the deployed solution is not practical as it needs information about the global optimum, which violates the black-box assumption of problems. To address this issue, it is suggested in [16] to replace the global optimum position $\mathbf{x}^{\star(t)}$ in (S-11) with the best-found position[2] $\mathbf{g}^{\star(t)}$.

Fu et al. [18] presented another definition that is independent of the knowledge of the global optimum or the best-found position:

$$\mathfrak{a}^{(t)}(\mathbf{s}) = \begin{cases} 0, & \text{if } f^{(t)}(\mathbf{s}) < \mu \\ 1, & \text{otherwise} \end{cases}, \tag{S-12}$$

where μ is a predefined fitness threshold used to evaluate the acceptability of $\mathbf{s}$. Therefore, a new solution must be chosen for deployment if the fitness value of the deployed solution in the new environment is worse than μ. In other words, (S-12) indicates that $\mathbf{s}$ can be kept as the deployed solution as long as its fitness value remains above μ after each environmental change. Equation (S-12) is used to evaluate the acceptability of solutions in other works such as [17, 29–31]. In [19], (S-12) was used to determine the acceptability of the deployed solution in $\mathrm{ROOT}_{\mathrm{G}}^{\mathrm{S}}$.

The definition of quality threshold in $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{M}}$ benchmark problems determines the acceptability of the deployed set of solutions, which is different from the aforementioned definitions that focus on the acceptability of a single deployed solution. In [2], a quality definition for changing or keeping the deployed set of solutions $\mathcal{P}$ for solving $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{M}}$ problems is used that is formulated as:

$$\mathfrak{a}^{(t)}(\mathcal{P}) = \begin{cases} 0, & \text{if } \exists \mathbf{x}_j \in \mathcal{P} : \dfrac{\left\| F^{(t-k)}(\mathbf{x}_j) - F^{(t)}(\mathbf{x}_j) \right\|}{\left\| F^{(t-k)}(\mathbf{x}_j) \right\|} > \mu \\ 1, & \text{otherwise} \end{cases}, \tag{S-13}$$

where the $(t-k)$th environment is when the current set of deployed solutions has been deployed, t is the index of the current environment, $F^{(t)}(\mathbf{x}_j)$ and $F^{(t-k)}(\mathbf{x}_j)$ are the objective function values of solution $\mathbf{x}_j \in \mathcal{P}$ in the objective space in the tth and $(t-k)$th environments, respectively, $\|\cdot\|$ calculates the $L2$-norm distance, and μ is a user defined acceptability threshold. Herein, a new set of solutions $\mathcal{P}$ must be deployed if $\mathfrak{a}^{(t)}(\mathcal{P}) = 0$.

2) Switching cost in ROOT benchmarks: In [30], switching cost is defined as the Euclidean distance between a candidate solution $\mathbf{x}$ and the deployed solution $\mathbf{s}$:

$$\mathfrak{c}(\mathbf{s}, \mathbf{x}) = \|\mathbf{s} - \mathbf{x}\|. \tag{S-14}$$

Note that the use of Euclidean distance as the switching cost is not configurable, making this definition unsuitable for generating problem instances with various degrees of switching costs. In addition, considering the dimensions and search ranges, the ratio between Euclidean distances and fitness values is not controllable. To address these issues, in [19], a configurable and more flexible definition of switching cost based on the Euclidean distance is used, which is formulated as:

$$\mathfrak{c}(\mathbf{s}, \mathbf{x}) = \omega \frac{\|\mathbf{s} - \mathbf{x}\|}{\sqrt{d}}, \tag{S-15}$$

where $\omega \geq 0$ is a weight parameter used for controlling the value of switching cost, and the Euclidean distance is divided

[2]In terms of the actual objective function value.

by $\sqrt{d}$ to make the switching cost value independent from the dimension number.

C. Discussion on ROOT benchmarks

In this section, we have reviewed the baseline functions of ROOT benchmark problems that have been modified or designed specifically for evaluating ROOT methods. We have also reviewed the components used in ROOT benchmark problems that define the requirements for changing or keeping deployed solutions. By adjusting the parameters of these components, researchers can change the quality threshold and switching cost, which affect the problem characteristics. For example, in (S-12), larger values of μ increase the problem's difficulty as the areas containing robust solutions shrink. However, setting these parameters (e.g., μ) to values that are too large or too small can change the problem's nature and make it behave similarly to other classes of optimization problems. For instance, a too-large μ value can result in lack of existing robust solutions in the search space. In such circumstances, the best approach would be to find and deploy the best solution in each environment, resulting in a TMO problem. Conversely, too small values of μ result in existing solutions that are acceptable over all $t_{\max}$ environments, causing the problem to behave like a robust optimization problem. A similar situation occurs for switching cost, where too small or large values can result in turning a $\mathrm{ROOT}_{\mathrm{G}}$ problem into a TMO or a $\mathrm{ROOT}_{\mathrm{Q}}$, respectively. Therefore, it is important for researchers using the ROOT benchmarks to carefully set these parameters in order to generate problem instances with the desired features. Another point to note is that the parameters controlling the quality threshold and switching cost in ROOT benchmarks remain constant over time, whereas in real-world problems, these parameters may change over time.

Another observation is that the majority of work in the field focuses on solving problems with *continuous* search spaces, and to the best of our knowledge, all of these benchmarks are artificial, i.e., they are not inspired by real-world problems. A few researchers in the field have worked on problems with *discrete* search spaces, including nonlinear dynamic stochastic optimization problems for stochastic energy management [32, 33] and dynamic customer location-allocation [34, 35]. These discrete problems were used in these studies as real-world-inspired benchmarks to evaluate $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{S}}$ methods. Note that these discrete benchmarks have not been specifically modified or designed for the field of ROOT.

S-II. ROOT PERFORMANCE INDICATORS

Performance indicators are specifically designed based on the class of ROOT problems. As stated in Section II, to date, no investigation has been conducted in the field on $\mathrm{ROOT}_{\mathrm{T}}^{\mathrm{M}}$ and $\mathrm{ROOT}_{\mathrm{G}}^{\mathrm{M}}$, and thus there is no performance indicator dedicated for them. Below, we review the performance indicators used for measuring the performance of the ROOT methods used for solving $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{S}}$, $\mathrm{ROOT}_{\mathrm{T}}^{\mathrm{S}}$, $\mathrm{ROOT}_{\mathrm{G}}^{\mathrm{S}}$, and $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{M}}$ problems.

A. $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{S}}$ performance indicators

Let $\mathcal{S} = \{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_{|\mathcal{S}|}\}$ be a sequence of deployed solutions found by a ROOT algorithm for a $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{S}}$ problem, where $|\mathcal{S}|$ is the total number of deployed solutions and $1 \leq |\mathcal{S}| \leq t_{\max}$. Assume that b_i is the environment number in which $\mathbf{s}_i \in \mathcal{S}$ is deployed and n_i is the number of environmental changes where $\mathbf{s}_i$ remains acceptable after them. In [12], the average error of each solution $\mathbf{s}_i \in \mathcal{S}$ during the environments that it has been (re)used is calculated by:

$$e_i = \frac{1}{n_i + 1} \sum_{t=b_i}^{b_i+n_i} \left| f^{(t)}(\mathbf{x}^{\star(t)}) - f^{(t)}(\mathbf{s}_i) \right|, \tag{S-16}$$

where $\mathbf{x}^{\star(t)}$ is the global optimum position in the tth environment. For each solution $\mathbf{s}_i$ that is robust, its sensitivity is calculated by:

$$\hat{s}_i = \sqrt{\frac{1}{n_i} \sum_{t=b_i}^{b_i+n_i} \left(\left| f^{(t)}(\mathbf{x}^{\star(t)}) - f^{(t)}(\mathbf{s}_i) \right| - e_i \right)^2}, \tag{S-17}$$

where $\mathbf{s}_i \in \mathcal{R}$ and $\mathcal{R} = \{\mathbf{s}_i \in \mathcal{S} | n_i > 0\}$ is the set of robust solutions ($\mathcal{R} \subseteq \mathcal{S}$). Equations (S-16) and (S-17) focus on a deployed solution $\mathbf{s}_i$. The average error of all deployed solutions $\mathcal{S}$ is calculated by:

$$e_{\mathrm{avg}} = \frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} e_i. \tag{S-18}$$

In addition, the average sensitivity of all robust solutions $\mathcal{R}$ is calculated by:

$$\hat{s}_{\mathrm{avg}} = \frac{1}{|\mathcal{R}|} \sum_{i \in \mathcal{R}} \hat{s}_i. \tag{S-19}$$

Although (S-18) and (S-19) were introduced for measuring the performance of $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{S}}$ methods, they focus on error and sensitivity, and do not take the robustness of the deployed solutions into account. Consequently, they may not really match for the definition of $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{S}}$ problems.

Later, Fu et al. [18] introduced a performance indicator that measures the performance of the algorithms based on the definition of survival time in (10):

$$\overline{S} = \frac{1}{t_{\max}} \sum_{t=1}^{t_{\max}} \mathfrak{s}\left(\mathbf{s}^{(t)}\right), \tag{S-20}$$

where $\overline{S}$ is the average survival time, $\mathfrak{s}\left(\mathbf{s}^{(t)}\right)$ is the number of successive environmental changes that the deployed solution $\mathbf{s}$ in the tth environment has been reused and remained acceptable after them. For example, for a solution $\mathbf{s}_i$ which is chosen in the fifth environment and remained acceptable at least until the eighth environment (i.e., $n_i = 3$), the values of $\mathfrak{s}\left(\mathbf{s}^{(5)}\right)$ to $\mathfrak{s}\left(\mathbf{s}^{(8)}\right)$ in (S-20) are $\{0, 1, 2, 3\}$, respectively. Therefore, we can also reformulate (S-20) as follows:

$$\overline{S} = \frac{1}{t_{\max}} \sum_{i=1}^{|\mathcal{S}|} \sum_{j=0}^{n_i} j. \tag{S-21}$$

B. $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{T}}$ *performance indicators*

To measure the performance of $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{T}}$ algorithms in maximizing the average fitness values of the deployed solutions over all environments, the following performance indicator was proposed in [18]:

$$\overline{A} = \frac{1}{t_{\max}} \sum_{t=1}^{t_{\max}} f^{(t)}\left(\mathbf{s}^{(t)}\right), \tag{S-22}$$

where $\overline{A}$ is the average fitness of the deployed solutions over all environments, and $\mathbf{s}^{(t)}$ is the deployed solution in tth environment. Although $\overline{A}$ is designed to measure $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{T}}$ algorithms, it has been also used to measure the average fitness of the deployed solutions in $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$ algorithms [17, 30].

C. $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{G}}$ *performance indicators*

In [19], to measure the performance of $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{G}}$, the following performance indicator is proposed:

$$\overline{G} = \frac{1}{t_{\max}} \sum_{t=1}^{t_{\max}} f^{(t)}\left(\mathbf{s}^{(t)}\right) - \mathfrak{c}\left(\mathbf{s}^{(t-1)}, \mathbf{s}^{(t)}\right), \tag{S-23}$$

where $\mathfrak{c}\left(\mathbf{s}^{(t-1)}, \mathbf{s}^{(t)}\right)$ calculates the switching cost between the deployed solution in the $(t-1)$th and tth environments, and $\overline{G}$ is the average gain over all environments. In (S-23), the switching cost shown by $\mathfrak{c}(\cdot)$ can be considered as a penalty value. Note that in environments where a new solution is not deployed (i.e., $\mathbf{s}^{(t-1)} = \mathbf{s}^{(t)}$), the switching cost is zero.

D. $\mathrm{ROOT}^{\mathrm{M}}_{\mathrm{Q}}$ *performance indicators*

Let $\widehat{\mathcal{P}} = \{\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_{|\widehat{\mathcal{P}}|}\}$ be a sequence of deployed POSs in a $\mathrm{ROOT}^{\mathrm{M}}_{\mathrm{Q}}$ problem. Assume that b_i is the environment number in which $\mathcal{P}_i$ is deployed and n_i is the number of environmental changes where $\mathcal{P}_i$ remains acceptable after them.

The average survival time is the main performance indicator used for measuring the performance of algorithms in tackling $\mathrm{ROOT}^{\mathrm{M}}_{\mathrm{Q}}$ problems [2–4] that is formulated as:

$$\overline{S} = \frac{1}{t_{\max}} \sum_{t=1}^{t_{\max}} \mathfrak{s}\left(\mathcal{P}^{(t)}\right), \tag{S-24}$$

where $\overline{S}$ is the average survival time, $\mathfrak{s}\left(\mathcal{P}^{(t)}\right)$ is the number of successive environmental changes that the set of deployed solutions $\mathcal{P}$ in the tth environment has been reused and remained acceptable after them. Note that the definition of average survival time in $\mathrm{ROOT}^{\mathrm{M}}_{\mathrm{Q}}$ is similar to that used for $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$ in (S-20).

In addition, to measure the quality of the deployed POS over time, two commonly used performance indicators in the field of multi-objective dynamic optimization –the generational distance (GD) and inverted generational distance (IGD) [36]– are modified for $\mathrm{ROOT}^{\mathrm{M}}_{\mathrm{Q}}$ in [2], which are called robust GD (RGD) and robust IGD (RIGD), respectively. RGD is calculates by:

$$\mathrm{RGD} = \frac{1}{|\widehat{\mathcal{P}}|} \sum_{i=1}^{|\widehat{\mathcal{P}}|} \max_{k \in \{b_i, \cdots, b_i + n_i\}} \mathrm{GD}\left(\mathcal{P}_i^{(k)}\right), \tag{S-25}$$

where $\mathrm{GD}\left(\mathcal{P}_i^{(k)}\right)$ calculates the generational distance [36] of $\mathcal{P}_i$ in the kth environment. Similarly, RIGD is calculated by:

$$\mathrm{RIGD} = \frac{1}{|\widehat{\mathcal{P}}|} \sum_{i=1}^{|\widehat{\mathcal{P}}|} \max_{k \in \{b_i, \cdots, b_i + n_i\}} \mathrm{IGD}\left(\mathcal{P}_i^{(k)}\right), \tag{S-26}$$

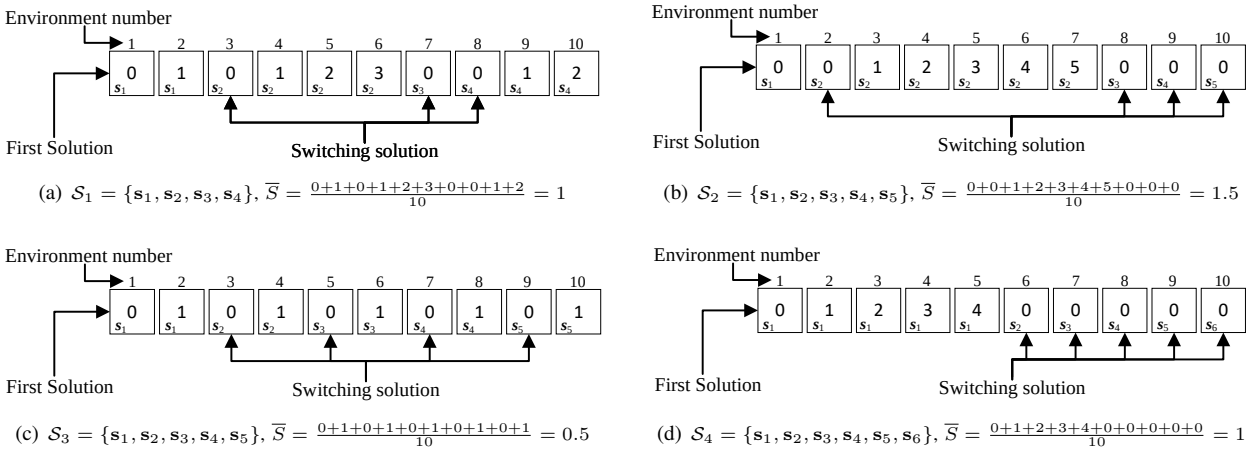
where $\mathrm{IGD}\left(\mathcal{P}_i^{(k)}\right)$ calculates the inverted generational distance [36] of $\mathcal{P}_i$ in the kth environment.

E. Discussion on ROOT performance indicators

Among the performance indicators used for measuring the performance of ROOT methods, the outputs of those that work based on the survival time metric –(S-20) and (S-24)– may be misleading. The average survival time is the most commonly used performance indicator to measure the performance of algorithms in solving $\mathrm{ROOT}_{\mathrm{Q}}$ problems [17, 24, 30]. However, using $\overline{S}$ values, we cannot determine which set of deployed solutions ($\mathcal{S}$ in $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$ and $\widehat{\mathcal{P}}$ in $\mathrm{ROOT}^{\mathrm{M}}_{\mathrm{Q}}$) by the algorithms is better in comparisons. To reveal the flaw of this performance indicator, we provide several examples in Fig. S-2, which compares four sets of deployed solutions $\mathcal{S}_{1,2,3,4}$ deployed in a $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$ problem. In Fig. 2(a) and 2(b), the average survival time is calculated for two sets of deployed solutions. Although $|\mathcal{S}_1| < |\mathcal{S}_2|$, the average survival time for $\mathcal{S}_2$ is better. However, with respect to the main goal of $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$ problems, which is to minimize the number of times the deployed solution is changed, $\mathcal{S}_1$ is better than $\mathcal{S}_2$. Consequently, the derivation from comparing the obtained average survival time values for $\mathcal{S}_1$ and $\mathcal{S}_2$ is contrary to the main objective of $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$, which is minimizing $|\mathcal{S}|$. This situation can also be observed when we compare the examples shown in Figs. 2(a) and 2(d). In these two figures, while the average survival time values obtained are identical, $|\mathcal{S}_4|$ is 50% worse than $|\mathcal{S}_1|$.

Another shortcoming of $\overline{S}$ can be revealed by comparing the examples illustrated in Figure 2(c) and 2(d). As can be seen, according to the calculated average survival time in these examples, $\mathcal{S}_4$ is better than $\mathcal{S}_3$. However, $|\mathcal{S}_3| < |\mathcal{S}_4|$. Moreover, all solutions in $\mathcal{S}_3$ are robust ($n_{1,2,3,4,5} > 0$), i.e., all of them had remained acceptable after an environmental change, while the majority of the deployed solutions in Fig. 2(d): solutions $\{\mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4, \mathbf{s}_5\} \in \mathcal{S}_4$ are not robust ($n_{2,3,4,5} = 0$). Finally, unlike the example depicted in Fig. 2(c), Fig. 2(d) suffers from successive changes in the deployed solutions in environments six to 10, which is undesirable in $\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$ problems.

Another important flaw of the average survival time is revealed by comparing the ratio between the obtained results in Figs. 2(a) to 2(d). It can be seen in these figures that the ratio between the results is unreliable for comparing algorithms. For example, the obtained result in Fig. 2(b) is three times larger than that of shown in Figure 2(c) while in both cases, the deployed solution is changed four times. Thus, the ratio between the obtained average survival time values by the algorithms is not reliable to show how much an algorithm is better than another.



(a) $\mathcal{S}_1 = \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4\}$, $\overline{S} = \frac{0+1+0+1+2+3+0+0+1+2}{10} = 1$

(b) $\mathcal{S}_2 = \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4, \mathbf{s}_5\}$, $\overline{S} = \frac{0+0+1+2+3+4+5+0+0+0}{10} = 1.5$

(c) $\mathcal{S}_3 = \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4, \mathbf{s}_5\}$, $\overline{S} = \frac{0+1+0+1+0+1+0+1+0+1}{10} = 0.5$

(d) $\mathcal{S}_4 = \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4, \mathbf{s}_5, \mathbf{s}_6\}$, $\overline{S} = \frac{0+1+2+3+4+0+0+0+0+0}{10} = 1$

Fig. S-2. Four examples of calculating average survival time by (S-20). Each square shows an environment. For each environment, the value of $\mathfrak{s}(\mathbf{s})$ in (S-20) is shown inside each square. In these examples, the problem has 10 environments, and $\mathcal{S}_{1,2,3,4}$ are the sets of deployed solutions over all environments in examples (a), (b), (c), and (d), respectively.

Note that (S-24), which is used for $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{M}}$, suffers from similar flaws of (S-20), which can be seen in circumstances similar to those provided in Fig. S-2 for $\mathrm{ROOT}_{\mathrm{Q}}^{\mathrm{S}}$.

REFERENCES

[1] D. Yazdani, R. Cheng, D. Yazdani, J. Branke, Y. Jin, and X. Yao, "A survey of evolutionary continuous dynamic optimization over two decades – part B," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 630–650, 2021.

[2] M. Chen, Y. Guo, H. Liu, and C. Wang, "The evolutionary algorithm to find robust pareto-optimal solutions over time," *Mathematical Problems in Engineering*, vol. 2015, 2015.

[3] Y. Guo, H. Yang, M. Chen, J. Cheng, and D. Gong, "Ensemble prediction-based dynamic robust multi-objective optimization methods," *Swarm and Evolutionary Computation*, vol. 48, pp. 156–171, 2019.

[4] Y. Guo, H. Yang, M. Chen, D. Gong, and S. Cheng, "Grid-based dynamic robust multi-objective brain storm optimization algorithm," *Soft Computing*, vol. 24, no. 10, pp. 7395–7415, 2020.

[5] M. Chen, Y. Guo, Y. Jin, S. Yang, D. Gong, and Z. Yu, "An environment-driven hybrid evolutionary algorithm for dynamic multi-objective optimization problems," *Complex & Intelligent Systems*, pp. 1–17, 2022.

[6] S. Biswas, S. Das, P. N. Suganthan, and C. A. C. Coello, "Evolutionary multiobjective optimization in dynamic environments: A set of novel benchmark functions," in *IEEE Congress on Evolutionary Computation*. IEEE, 2014, pp. 3192–3199.

[7] M. Farina, K. Deb, and P. Amato, "Dynamic multiobjective optimization problems: test cases, approximations, and applications," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 5, pp. 425–442, 2004.

[8] C.-K. Goh and K. C. Tan, "Evolutionary multi-objective optimization in uncertain environments," *Issues and Algorithms, Studies in Computational Intelligence*, vol. 186, pp. 5–18, 2009.

[9] M. Helbig and A. P. Engelbrecht, "Benchmarks for dynamic multi-objective optimisation algorithms," *ACM Computing Surveys (CSUR)*, vol. 46, no. 3, pp. 1–39, 2014.

[10] S. Jiang, J. Zou, S. Yang, and X. Yao, "Evolutionary dynamic multi-objective optimisation: A survey," *ACM Computing Surveys*, vol. 55, no. 4, pp. 1–47, 2022.

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

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

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

[14] J. Branke, *Evolutionary optimization in dynamic environments*. Springer Science & Business Media, 2012, vol. 3.

[15] R. W. Morrison and K. A. D. Jong, "A test problem generator for non-stationary environments," in *IEEE Congress on Evolutionary Computation*, vol. 3. IEEE, 1999, pp. 2047–2053.

[16] Y. Jin, K. Tang, X. Yu, B. Sendhoff, and X. Yao, "A framework for finding robust optimal solutions over time," *Memetic Computing*, vol. 5, no. 1, pp. 3–18, 2013.

[17] D. Yazdani, T. T. Nguyen, and J. Branke, "Robust optimization over time by learning problem space characteristics," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 1, pp. 143–155, 2019.

[18] H. Fu, B. Sendhoff, K. Tang, and X. Yao, "Finding robust solutions to dynamic optimization problems," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2013, pp. 616–625.

[19] D. Yazdani, J. Branke, M. N. Omidvar, T. T. Nguyen, and X. Yao, "Changing or keeping solutions in dynamic optimization problems with switching costs," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2018, pp. 1095–1102.

[20] D. Yazdani, M. N. Omidvar, J. Branke, T. T. Nguyen, and X. Yao, "Scaling up dynamic optimization problems: A divide-and-conquer approach," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 1, pp. 1–15, 2020.

[21] D. Yazdani, M. N. Omidvar, R. Cheng, J. Branke, T. T. Nguyen, and X. Yao, "Benchmarking continuous dynamic optimization: Survey and generalized test suite," *IEEE Transactions on Cybernetics*, vol. 52, no. 5, pp. 3380–3393, 2022.

[22] D. Yazdani, J. Branke, M. N. Omidvar, X. Li, C. Li, M. Mavrovouniotis, T. T. Nguyen, S. Yang, and X. Yao, "IEEE CEC 2022 competition on dynamic optimization problems generated by generalized moving peaks benchmark," *arXiv: 2106.06174*, 2021.

[23] D. Yazdani, D. Yazdani, J. Branke, M. N. Omidvar, Amir H. Gandomi, and X. Yao, "Robust optimization over time by estimating robustness of promising regions," *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 3, pp. 657–670, 2022.

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

731–745, 2015.
[25] C. Li, S. Yang, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H.-G. Beyer, and P. N. Suganthan, "Benchmark generator for cec'2009 competition on dynamic optimization," Center for Computational Intelligence, Tech. Rep., 2008.
[26] J. Branke, *Evolutionary Optimization in Dynamic Environments*. Springer-Verlag Berlin Heidelberg, 2002.
[27] J. Brimberg, P. Hansen, N. Mladenovic, and S. Salhi, "A survey of solution methods for the continuous location-allocation problem," *International Journal of Operations Research*, vol. 5, no. 1, pp. 1 – 12, 2008.
[28] P. Novoa-Hernández, D. A. Pelta, and C. C. Corona, "Approximation models in robust optimization over time-an experimental study," in *IEEE Congress on Evolutionary Computation*. IEEE, 2018, pp. 1–6.
[29] Y. Huang, Y. Jin, and K. Hao, "Decision-making and multi-objectivization for cost sensitive robust optimization over time," *Knowledge-Based Systems*, p. 105857, 2020.
[30] Y. Huang, Y. Ding, K. Hao, and Y. Jin, "A multi-objective approach to robust optimization over time considering switching cost," *Information Sciences*, vol. 394, pp. 183–197, 2017.
[31] D. Yazdani, T. T. Nguyen, J. Branke, and J. Wang, "A new multi-swarm particle swarm optimization for robust optimization over time," in *Applications of Evolutionary Computation*, G. Squillero and K. Sim, Eds. Springer International Publishing, 2017, pp. 99–109.
[32] Y. Liu and H. Liang, "A ROOT approach for stochastic energy management in electric bus transit center with PV and ESS," in *IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.
[33] ——, "A three-layer stochastic energy management approach for electric bus transit centers with PV and energy storage systems," *IEEE Transactions on Smart Grid*, vol. 12, no. 2, pp. 1346–1357, 2020.
[34] R. Ankrah, B. Lacroix, J. McCall, A. Hardwick, and A. Conway, "Introducing the dynamic customer location-allocation problem," in *IEEE Congress on Evolutionary Computation*. IEEE, 2019, pp. 3157–3164.
[35] R. Ankrah, B. Lacroix, J. McCall, A. Hardwick, A. Conway, and G. Owusu, "Racing strategy for the dynamic-customer location-allocation problem," in *IEEE Congress on Evolutionary Computation*. IEEE, 2020, pp. 1–8.
[36] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

Nomenclature

Notation	Description
ROOT	Robust optimization over time
DOP	Dynamic optimization problem
TMO	Tracking the moving optimum
$\mathrm{ROOT}^{\mathrm{S}}$	ROOT problems in which there is one and only one deployed solution at each point in time.
$\mathrm{ROOT}^{\mathrm{M}}$	ROOT problems in which there are multiple deployed solutions.
$\mathrm{ROOT}_{\mathrm{Q}}$	ROOT problems with a quality threshold for determining the acceptability of the deployed solution(s).
$\mathrm{ROOT}_{\mathrm{T}}$	ROOT problems with a time-window-based temporal threshold for deploying a new solution(s).
$\mathrm{ROOT}_{\mathrm{G}}$	ROOT problems in which the gain of switching the deployed solution(s) alongside its/their acceptability are considered
$\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{Q}}$	Combination of $\mathrm{ROOT}_{\mathrm{Q}}$ and $\mathrm{ROOT}^{\mathrm{S}}$.
$\mathrm{ROOT}^{\mathrm{M}}_{\mathrm{Q}}$	Combination of $\mathrm{ROOT}_{\mathrm{Q}}$ and $\mathrm{ROOT}^{\mathrm{M}}$.
$\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{T}}$	Combination of $\mathrm{ROOT}_{\mathrm{T}}$ and $\mathrm{ROOT}^{\mathrm{S}}$.
$\mathrm{ROOT}^{\mathrm{M}}_{\mathrm{T}}$	Combination of $\mathrm{ROOT}_{\mathrm{T}}$ and $\mathrm{ROOT}^{\mathrm{M}}$.
$\mathrm{ROOT}^{\mathrm{S}}_{\mathrm{G}}$	Combination of $\mathrm{ROOT}_{\mathrm{G}}$ and $\mathrm{ROOT}^{\mathrm{S}}$.
$\mathrm{ROOT}^{\mathrm{M}}_{\mathrm{G}}$	Combination of $\mathrm{ROOT}_{\mathrm{G}}$ and $\mathrm{ROOT}^{\mathrm{M}}$.
EA	Evolutionary algorithm
EDOA	Evolutionary dynamic optimization algorithm
PSO	Particle swarm optimization
DE	Differential evolution
MPB	Moving peaks benchmark
GMPB	Generalized moving peaks benchmark
mMPB	Modified moving peaks benchmark
POS	Pareto-optimal solutions
f	Objective function
t	Time index in dynamic optimization problems. t shows the environment number.
$\mathbf{x}$	A solution in the search space. $\mathbf{x}$ is a d-dimensional vector where d is the problem dimensionality.
$t_{\max}$	Maximum time index, which shows the number of environments.
α	A set of time-dependent control parameters of the objective function.
$\mathbf{s}_i$	ith deployed solution
b_i	Index of the environment in which $\mathbf{s}_i$ is deployed.
n_i	The number of environmental changes beyond which $\mathbf{s}_i$ remained acceptable.
l	Total number of deployments
$\mathcal{S}$	Set of deployed solutions, where $\mathcal{S} = \{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_l\}$
$\mathfrak{a}^{(t)}(\cdot)$	It is a binary function that checks the acceptability of a solution in the tth environment. $\mathfrak{a}^{(t)}(\mathbf{s}_i)$ returns one if $\mathbf{s}_i$ is acceptable in the tth environment, and otherwise, it returns zero.
t_{win}	Length of time window in $\mathrm{ROOT}_{\mathrm{T}}$
$\mathcal{T}$	Set of time windows
F	Multi-objective problem
$\hat{m}$	Number of objectives in a multi-objective problem
$\mathcal{P}$	A deployed set of solutions in $\mathrm{ROOT}^{\mathrm{M}}$
$\widehat{\mathcal{P}}$	Set of deployed sets of solutions in $\mathrm{ROOT}^{\mathrm{M}}$, where $\widehat{\mathcal{P}} = \{\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_l\}$
$\breve{f}$	Substitute objective function
f'	Approximation function
f''	Prediction function
p	Number of considered previous environments in estimating robustness of a solution/POS
q	Number of considered future environments in estimating robustness of a solution/POS
μ	Quality threshold for evaluating the acceptability of the deployed solution or POS in $\mathrm{ROOT}_{\mathrm{Q}}$ and $\mathrm{ROOT}_{\mathrm{G}}$.
$\tau_i^{(t)}$	Fitness fluctuation observed by the ith sub-population at the tth environmental change.
$\bar{\tau}_i$	Average of previous τ_i values
$\mathbf{g}_i^{*(t)}$	The best-found position by the ith sub-population in the tth environment.
$\mathfrak{r}(\cdot)$	A binary function that estimates the robustness of the promising region covered by a sub-population in the upcoming environment.
$\mathcal{C}$	Set of robust promising regions, i.e., the ones whose $\mathfrak{r}\left(\mathbf{g}_i^{*(t)}, \bar{\tau}_i\right) = 1$.
$c^{*(t)}$	Chosen promising region in the tth environment from which a solution is chosen for deployment.
d	Dimension number
$s_i'^{(t)}$	Estimated relocation length of the promising region covered by the ith sub-population at the tth environmental change
$h_i^{(t)}$	Estimated height difference of the promising region covered by the ith sub-population at the tth environmental change

$\bar{s}'_i$	Estimated relocation length of the promising region covered by the ith sub-population
$\bar{h}'_i$	Estimated height difference of the promising region covered by the ith sub-population
$\bar{s}'_{\max}$	Largest $\bar{s}'_{i\in\mathcal{C}}$
$\bar{h}'_{\max}$	Largest $\bar{h}'_{i\in\mathcal{C}}$
$\mathfrak{c}(\mathbf{s},\mathbf{x})$	A function that calculates the switching cost between the deployed solution $\mathbf{s}$ and a candidate solution $\mathbf{x}$.
$\mathbf{x}^{\star(t)}$	The global optimum position in the tth environment.
$\mathbf{g}^{\star(t)}$	The best-found position by the optimization algorithm in the tth environment.
δ	Threshold of maximum tolerated quality difference between $\mathbf{s}$ and $\mathbf{x}^{\star(t)}$ in $\mathrm{ROOT_Q}$
m	Defines the number of promising regions in the benchmark generators.
$h_i^{(t)}$	Height of the ith promising region in the tth environment in MPB-based benchmarks
$w_i^{(t)}$	Width of the ith promising region in the tth environment in MPB-based benchmarks
$\mathbf{c}_i^{(t)}$	Center/summit position of the ith promising region in the tth environment in MPB-based benchmarks
$\mathrm{N}(0,1)$	A random number drawn from a Gaussian distribution with mean 0 and variance 1
λ	Correlation coefficient in in MPB-based benchmarks
$\tilde{h}_i$	Change severity parameter of height of the ith promising region in MPB-based benchmarks
$\tilde{w}_i$	Change severity parameter of width of the ith promising region in MPB-based benchmarks
$\tilde{s}_i$	Shift severity parameter of the ith promising region in MPB-based benchmarks
τ	Irregularity parameter in GMPB
η	Irregularity parameter in GMPB
ω	Weight parameter used in calculating switching cost
$\mathbb{R}$	Reals
$\mathbb{T}$	Transformation function in GMPB
$\mathbf{R}$	Rotation matrix in GMPB
$\mathbf{W}$	A $d\times d$ diagonal matrix whose diagonal elements show the width of a promising region in different dimensions (used in GMPB).
e	Average error of each deployed solution $\mathbf{s}_i\in\mathcal{S}$ during the environments that it has been (re)used.
$\mathcal{R}$	Set of robust solutions, i.e., $\mathcal{R}=\{\mathbf{s}_i\in\mathcal{S}\|n_i>0\}$ and $\mathcal{R}\subseteq\mathcal{S}$.
e_i	Average error of the deployed solution $\mathbf{s}_i\in\mathcal{S}$
e_{avg}	Average error of all deployed solutions $\in\mathcal{S}$
$\hat{s}_i$	Sensitivity of the robust deployed solution $\mathbf{s}_i\in\mathcal{R}$
$\hat{s}_{\mathrm{avg}}$	Average sensitivity of all robust deployed solution $\in\mathcal{R}$
$\mathfrak{s}\left(\mathbf{s}^{(t)}\right)$	$\mathfrak{s}(\cdot)$ is a function that shows the number of successive environmental changes that the deployed solution $\mathbf{s}$ in the tth environment has been reused.
$\overline{S}$	Average survival time
$\overline{A}$	Average fitness
$\overline{G}$	Average gain
GD	Generational distance
IGD	Inverted generational distance
RGD	Robust generational distance
RIGD	Robust inverted generational distance