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

A framework for finding robust optimal solutions over time

Yaochu Jin, Ke Tang, Xin Yu, Bernhard Sendhoff, Xin Yao

2013

Preprint:

This is an accepted article published in Memetic Computing. The final authenticated version is available online at: https://doi.org/[DOI not available]

Memetic Comp. (2013) 5:3–18
DOI 10.1007/s12293-012-0090-2

REGULAR RESEARCH PAPER

A framework for finding robust optimal solutions over time

Yaochu Jin · Ke Tang · Xin Yu · Bernhard Sendhoff · Xin Yao

Received: 18 June 2012 / Accepted: 3 August 2012 / Published online: 4 September 2012
© Springer-Verlag 2012

Abstract Dynamic optimization problems (DOPs) are those whose specifications change over time, resulting in changing optima. Most research on DOPs has so far concentrated on tracking the moving optima (TMO) as closely as possible. In practice, however, it will be very costly, if not impossible to keep changing the design when the environment changes. To address DOPs more practically, we recently introduced a conceptually new problem formulation, which is referred to as robust optimization over time (ROOT). Based on ROOT, an optimization algorithm aims to find an acceptable (optimal or sub-optimal) solution that changes slowly over time, rather than the moving global optimum. In this paper, we propose a generic framework for solving DOPs using the ROOT concept, which searches for optimal solutions that are robust over time by means of local fitness approximation and prediction. Empirical investigations comparing a few representative TMO approaches with an instantiation of the proposed framework are conducted on a number of test problems to demonstrate the advantage of the proposed framework in the ROOT context.

Keywords Robust optimisation over time (Root) · Dynamic optimisation · Evolutionary algorithms · Particle swarm optimisation · Fitness approximation

Y. Jin · K. Tang · X. Yu · X. Yao
Nature Inspired Computation and Applications Laboratory (NICAL), School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, Anhui, China

K. Tang
e-mail: ketang@ustc.edu.cn

Y. Jin (✉)
Department of Computing, University of Surrey, Guildford, Surrey GU2 7XH, UK
e-mail: yaochu.jin@surrey.ac.uk

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

X. Yao
Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, UK
e-mail: x.yao@cs.bham.ac.uk

1 Introduction

Most real-world optimization problems are often subject to changing environments. Changes in an optimization problem can involve variations in the objective functions, decision variables, environmental parameters, as well as the constraints. The number of objectives, decision variables and constraints may also vary from time to time during the optimization. Such optimization problems are termed as dynamic optimization problems (DOPs) [1,2].

Population-based optimization algorithms (POAs), such as evolutionary algorithms (EAs), are considered to be well suited for solving DOPs [3]. By far, most research on dynamic optimization has focused on tracking moving optima (TMO) [2,4–8], where, whenever a change occurs, the new global optimal solution is targeted. Such approaches are theoretically possible, but practically of less interest, since in most real-world scenarios, it is impractical to change the design (or a plan) in use constantly. In our previous work [9], we pointed out a number of limitations of TMO approaches to DOPs, including the difficulty in relocating the new optima in severely or quickly changing environments, the impracticality of using the new solutions due to too many human operations [10] or limited resources such as time and costs, and the risk of being deceived by the "time-linkage" problem [11]. Meanwhile, we proposed to

find robust optimal solutions over time, which was referred to as robust optimization over time (ROOT) [9]. According to ROOT, the goal of optimization is to find a sequence of acceptable solutions, which can be optimal or sub-optimal, instead of the global optimum in any given environment. One assumption is that when the environment changes, the old solution can still be used in the new environment as long as its quality is acceptable. The criterion for an acceptable optimal solution is problem-specific and can be pre-defined by the user.

The main new contributions of this paper are to propose a generic framework for solving DOPs based on the ROOT concept. The proposed framework consists of a population-based optimization algorithm (POA), a database, a fitness approximator and a fitness predictor. A definition for robustness over time, which is the essence of the framework, is also proposed. Simply put, a solution's robustness over time is estimated by both its past and future performances. To demonstrate the feasibility of the framework, an instantiation of the generic framework is given, where a radial-basis-function (RBF) network [12] is adopted as the local approximator [13,14] of past fitness values and the autoregressive (AR) model [15] for predicting future performance. The performance of the instantiation as well as a few representative TMO approaches is examined on nine DOP test problems that belong to three categories of ROOT problems, including mMPB [9] and two other benchmark problems proposed in this paper. Finally, we demonstrate the effectiveness of the framework and empirically analyze the performance of the implemented prototype on the above test problems.

2 Related work

In this section, we present a brief introduction to research on robust optimization and dynamic optimization, both are closely related to ROOT. More comprehensive surveys on dynamic optimization and robust optimization can be found in [1,2,16,17]. Without loss of generality, we assume the following maximization problem is in consideration:

$$\text{Maximize:} \quad F(\vec{x}) = f(\vec{x}, \vec{\alpha}), \tag{1}$$

where f is the objective function, $\vec{x}$ is a vector of design variables and $\vec{\alpha}$ is the vector of environmental parameters.

2.1 Robust optimization

One widely used definition of robust solutions is a solution's expected performance over all possible disturbances. Thus the resultant fitness of Eq. 1 is

$$F(\vec{x}) = \int_{-\infty}^{+\infty} f(\vec{x} + \vec{\delta}, \vec{\alpha}) p(\vec{\delta}) d\vec{\delta}, \tag{2}$$

or

$$F(\vec{x}) = \int_{-\infty}^{+\infty} f(\vec{x}, \vec{\alpha} + \vec{\xi}) p(\vec{\xi}) d\vec{\xi}, \tag{3}$$

where $\vec{\delta}$ and $\vec{\xi}$ are terms for describing noise in the design variables and environmental parameters, respectively, and $p(\vec{\delta})$ and $p(\vec{\xi})$ are their probability density functions.

In practice, it is often impossible to analytically calculate the expected values of Eqs. 2 and 3. An intuitive alternative is to estimate them using Monte Carlo integration by sampling over a number of realizations of $\vec{\delta}$ and $\vec{\xi}$ [18–20]. To provide a more reliable estimation in the Monte Carlo integration while avoiding additional fitness evaluations [21,22], surrogate models [23,24] have been employed to substitute the original fitness function [25,29,26]. Surrogates have also been studied in memetic algorithms [27,28]. Ong et al. [29] employed the radial basis functions (RBFs) as the local surrogate models and incorporated them into a Baldwinian trust-region framework to estimate a solution's worst case performance. The experimental results verified the effectiveness of the above work on robust optimization [25,29]. Most recently, a probabilistic model has been developed in [30] for choosing a surrogate that is expected to bring about highest fitness improvement among a number of given surrogates based on the work in [31,32]. However, dynamic optimization problems defined in this paper have not been not considered.

2.2 TMO approach to DOPs

In a DOP, the fitness function is deterministic at any given time instant, but dependant on time t. Hence the resultant fitness of Eq. 1 takes the form

$$F(\vec{x}) = f(\vec{x}, \vec{\alpha}(t)), \tag{4}$$

where t is the time index with $t \in [0, T_{end}]$ (T_{end} is the life cycle of the problem, $T_{end} > 0$). Now the problem parameters $\vec{\alpha}(t)$ are changing over time, and the objective of TMO is to maximize the function defined in Eq. 4 at any time instance.

In practice, an assumption of "small to medium" degree of environmental changes is often made so that "tracking" makes sense [33]. To enhance the ability of POAs to track a moving optimum, two main strategies have been adopted. One is to make use of historical information [33–38] and the other is to re-introduce or maintain the population diversity [4,39–44].

2.3 Robust optimization over time

In the context of ROOT, the problem to be solved is similar to that defined in Eq. 4. However, the goal here is to

find solutions whose quality is acceptable over a certain time interval, although they must not be the global optima at any time instant. From this perspective, the target is to optimize the expected fitness over time, i.e. [9],

$$F(\vec{x}) = \int_{t_0}^{t_0+T} \int_{-\infty}^{+\infty} f(\vec{x}, \vec{\alpha}(t)) p(\vec{\alpha}(t)) d\vec{\alpha}(t) dt, \tag{5}$$

where $p(\vec{\alpha}(t))$ is the probability density function of $\vec{\alpha}(t)$ at time t, T is the length of the time interval, t_0 is a given starting time. If we know the dynamics of the problem parameters, e.g., $\vec{\alpha}(t)$ is determined by

$$\vec{\alpha}(t) = g(\vec{\alpha}(0), \Delta\vec{\alpha}, t), \tag{6}$$

where $\Delta\vec{\alpha}$ is the change of α between two successive time instants.

From Eqs. 2–4 and 6, we can find that robustness in ROOT not only takes into account uncertainties in the decision space and parameter space, but also the effect of these uncertainties in the time domain. Our definition has also taken into account the "time-linkage" problems [11], where the optimal decision in the long run is deceived by keeping finding the global optimum of the problem at any given time.

3 Discrete-time dynamic optimization problems

3.1 Problem formulation

In this section, we discuss robustness over time in the context of a class of discrete-time dynamic optimization problems (DTDOPs), together with the formal definitions of TMO and ROOT. We also briefly analyze the characteristics of ROOT on DTDOPs.

In Eq. 4, the time t is normally assumed to be discrete and is measured in function evaluations in the field of DOPs. Here we consider a class of dynamic optimization problems that are most widely studied in the literature: The parameters of the problem change over time with stationary periods between changes. In other words, $\vec{\alpha}(t)$ does not have to change continuously over time, and thus within T_{end} there may be a sequence of $l = \lceil T_{end}/\tau \rceil$ different instances of the same problem (the problem with parameters $\langle \vec{\alpha}_1, \vec{\alpha}_2, \ldots, \vec{\alpha}_l \rangle$), where $1/\tau$ is the frequency of the change and usually remains constant but could also be time-variant. Therefore, the problem can be re-formulated as:

$$\langle f(\vec{x}, \vec{\alpha}_1), f(\vec{x}, \vec{\alpha}_2), \ldots, f(\vec{x}, \vec{\alpha}_l) \rangle. \tag{7}$$

We call this category of problems DTDOPs.

For DTDOPs defined in Eq. 7, TMO means to find the global optimal solution S_i^* within the i-th time interval T_i for the problem $f(\vec{x}, \vec{\alpha}_i)$ where $i = 1, 2, \ldots, l$. Thus the task is to find a sequence of corresponding global optimal solutions $\langle S_1^*, S_2^*, \ldots, S_l^* \rangle$.

3.2 Definition of ROOT in DTDOPs

The task of ROOT is to find a sequence of acceptable solutions that are robust over time. Formally, for the DTDOP defined in Eq. 7, the goal is to find a sequence of solutions $\langle S_1, S_2, \ldots, S_k \rangle$, and a solution S_i $(i = 1, 2, \ldots, k)$ is said to be robust if its performance is acceptable for at least two consecutive time intervals. Thus we have $1 \leq k \leq l$. Note that $k = l$ means that no such robust solutions exist and for every changed environment, a new solution must be found. On the other hand, the ideal situation occurs when $k = 1$, where only one solution is needed for all environments, and thus there is no need to search for a new solution when the environment changes.

Consider the dynamic problem defined above in the time interval $[t_L, t_U] \subseteq [0, T_{end}]$ with a simple additive dynamics between two successive changes, i.e., let $l_L = \lceil t_L/\tau \rceil$ and $l_U = \lceil t_U/\tau \rceil$, we obtain the problem

$$\langle f(\vec{x}, \vec{\alpha}_{l_L}), f(\vec{x}, \vec{\alpha}_{l_L+1}), \ldots, f(\vec{x}, \vec{\alpha}_{l_U}) \rangle, \tag{8}$$

with the dynamics

$$\vec{\alpha}_i = \vec{\alpha}_{i-1} + \Delta\vec{\alpha}, \tag{9}$$

where $i = l_L + 1, l_L + 2, \ldots, l_U$. Recall that f is to be maximized.

The dynamics parameter $\Delta\vec{\alpha}$ is regarded as a random variable obeying a certain distribution such as a Gaussian distribution or a uniform distribution. Here the parameters of the distribution functions (e.g., the mean and the standard deviation of the Gaussian distribution function) are assumed to be time-invariant. In this case, two suggested measures [9] are given below to estimate the robustness in ROOT:

$$F(\vec{x}; l_L, l_U) = \frac{1}{l_U - l_L + 1} \sum_{i=1}^{l_U - l_L + 1} \int f(\vec{x}, \vec{\alpha}_{l_L} + (i-1)\Delta\vec{\alpha}) p(\Delta\vec{\alpha}) d\Delta\vec{\alpha}, \tag{10}$$

and

$$D(\vec{x}; l_L, l_U) = \frac{1}{l_U - l_L + 1} \sum_{i=1}^{l_U - l_L + 1} \int (f(\vec{x}, \vec{\alpha}_{l_L} + (i-1)\Delta\vec{\alpha}) - F(\vec{x}; l_L, l_U))^2 p(\Delta\vec{\alpha}) d\Delta\vec{\alpha}, \tag{11}$$

where $p(\Delta\vec{\alpha})$ is the probability density function of $\Delta\vec{\alpha}$. It can be seen that F actually measures the average performance over all environments within the time interval $[t_L, t_U]$, and

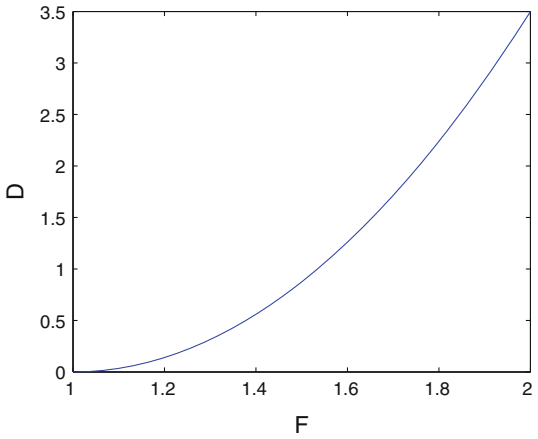

Fig. 1 Pareto front of the example function in Eq. 12 aiming for maximizing F in Eq. 13 and minimizing D in Eq.14. The *curve* represents the non-dominated solutions of the multi-objective optimization problem

D measures the degree to which the performance varies when the environment changes.

Taking the following scalar objective function to be maximized as an example:

$$f(x, \alpha) = \alpha - (\alpha - 1)x^2, \quad \alpha \in \mathbb{R}, x \in \mathbb{R}, \tag{12}$$

where α changes according to Eq. 9 and $\Delta\alpha$ is assumed to be normally distributed $\Delta\alpha \sim N(0, 1)$. In the time interval $[t_L, t_U]$, according to Eqs. 10 and 11, the two measures can be calculated as follows:

$$F = \alpha_{l_L} - (\alpha_{l_L} - 1)x^2, \tag{13}$$

and

$$D = \frac{(l_U - l_L)[2(l_U - l_L) + 1]}{6}(1 - x^2)^2. \tag{14}$$

From Eqs. 13 and 14, we can see that if $\alpha_{l_L} \leq 1$, maximizing F and minimizing D are not conflicting goals. However, if $\alpha_{l_L} > 1$, maximizing F and minimizing D represent conflicting objectives, and hence we are dealing with *multi-objective robust optimization*. Using the goals in Eqs. 13 and 14 (as an example Pareto front, here we set $\alpha_{l_L} = 2$ and $l_U - l_L = 3$), the Pareto front of the example function in Eq. 12 is shown in Fig. 1.

4 Population-based algorithms (POAs) for robust optimization over time

4.1 The framework

We propose a framework for solving ROOT using POAs, as they have been widely used and shown successful in the TMO approach to DOPs.

From the robustness measure defined in Eqs. 10 and 11, we can conceive that to estimate a solution's performance over a time interval, we must take into account its past and future performance. If we assume the change in performance is not random, both past and future performance can be estimated from historical data collected during optimization. Furthermore, a local approximator is needed to estimate the past fitness values of a solution if they cannot be directly retrieved from the database. Thus, our framework for solving ROOT problems should consist of an optimizer (an POA, e.g., a genetic algorithm or a particle swarm optimization algorithm), a database, an approximator and a predictor.

4.1.1 Approximator and predictor

When the optimization problem changes, we can employ an approximation model constructed with historical data to estimate a solution's performance in a previous time instant. An approximate model, often known as meta-model or surrogate, can also be very helpful when fitness evaluations are highly time-consuming as in many real-world applications [23,46].In this work, a local approximation model is constructed using its neighboring historical data in the database to estimate a solution's past performance. Many models, such as radial-basis-functions (RBFs) [29,14], polynomial functions [13,25], kriging models [45] can be chosen. For an overview of using fitness approximation in EAs, the reader is referred to [2,46].

By contrast, the task of the predictor is to estimate a solution's future performance. To this end, an approximation model must be constructed based on the solution's past performances that can either directly retrieved from the database or predicted using the local approximator introduced in the previous section. Any time series prediction techniques can be adopted for the predictor, such as autoregressive (AR) models [15], support vector machines (SVMs) [47], and models used for approximation mentioned in the previous subsection.

4.1.2 Database

The database is used to store historical data, based on which the local approximator and the predictor are constructed. It starts without any records and collects data when the POA is running. In each generation, each candidate's location in the search space, its fitness value and the associated time instant are stored into the database.

4.2 Estimation of robustness over time

From the previous sections, we can see that the critical issue in solving ROOT problems is to estimate the robustness over time as defined in Eqs. 10 and 11 of a solution. We limit our discussions to estimating the F measure in this paper.

Generally, the integral part of F is not easy to estimate since in real-world applications there is little prior knowledge

about the problem and the uncertainty. However, if we know the performance of a solution in several time steps, we can compute its average performance during a certain time interval and take this average performance as the solution's robustness over the considered time interval.

In other words, at time step l_0, we estimate the robustness of a solution $\vec{x}$ as follows

$$\hat{F}(\vec{x}; l_0) = \sum_{l=l_0-p}^{l_0+q} \hat{f}(\vec{x}, \vec{\alpha}_l), \tag{15}$$

where $\hat{F}$ actually looks backward to p past values and forward to q future values of a solution. Note that the approximated fitness $\hat{f}$ is used instead of the real fitness value, because the estimation of a solution's robustness should not introduce extra fitness evaluations. In this way, it is guaranteed that the problem being solved will not change again during the process of estimation.

The pseudo-code of the proposed algorithm, a $(p+q)$-POA, for ROOT is described in Algorithm 1, where g is the index of generation, l is the index of environment, 'DB', 'LA', and 'Pre' denotes the database, the local approximator, and the predictor, respectively. $(p+q)$-POA starts with an empty database and collects data during the run. In each generation, it operates like an ordinary POA except that the fitness now is an estimated F value.

4.3 Requirements on estimators

In order to guarantee an acceptable performance of the framework, the estimator should be unbiased. However for a POA that needs rank-based information to drive the search, a low standard deviation of the estimation error is more important than the mean as long as the biases are consistent on different points [25]. For example, if there is an estimator whose estimation error has a probability distribution with zero standard deviation, the search result of an POA with rank based selection will be the same, no matter whether the search is based on the estimator or on the real fitness function. For a PSO algorithm, since the updates of the personal best information and the global best information are all based on the comparisons between fitness values, fitness approximation errors will not change the locations of the attraction to the population.

Algorithm 1 $(p+q)$-POA – A population-based optimization algorithm for ROOT

```
g ← 0
l ← 0
initialize population P(0)
evaluate every individual in P(0) with real f
add (indi, 0) to DB where indi ∈ P(0)
repeat
  if the environment changes then
    l ← l + 1
  end if
  P'(g) ← select(P(g))
  P''(g) ← algorithmOperators(P'(g))
  evaluate every individual in P''(g) with real f
  add (indi, l) to DB where indi ∈ P''(g)
  estimate F of all P''(g) individuals via LA(p) and Pre(q)
  assign the estimated F value to the fitness of every individual in P''(g)
  P(g+1) ← survival(P(g) ⋃ P''(g))
  g ← g + 1
until termination criteria met
```

5 An instantiation of the framework

As an instantiation of the framework, we adopt a radial-basis-function (RBF) model [12] as the local approximator and an autoregressive (AR) model [15] as the predictor. Since AR is a linear model, we also employ a nonlinear RBF model as the predictor for comparison.

5.1 Radial-basis-function model

An RBF approximation model can be defined as follows:

$$y(\vec{x}_k) = \sum_{i=1}^{n_c} \omega_i \phi(\|\vec{x}_k - \vec{c}_i\|_2), \tag{16}$$

where $\{\vec{x}_k, y(\vec{x}_k), k = 1, 2, \ldots, n_s\}$ is the training dataset, $\vec{x}_k \in \mathbb{R}^d$ is the k-th input vector, $y(\vec{x}_k)$ is the k-th output, n_c is the number of centers, $\vec{c}_i$ is the i-th center, $\phi(\cdot) : \mathbb{R}^d \to \mathbb{R}$ is a radial basis kernel, $\|\cdot\|_2$ represents the Euclidean norm, and $\vec{\omega} = \{\omega_1, \omega_2, \ldots, \omega_{n_c}\}^T \in \mathbb{R}^{n_c}$ denotes the weight vector.

In this work, we adopt a linear kernel for the RBF model, which takes the following form:

$$y(\vec{x}_k) = \sum_{i=1}^{n_c} \omega_i \|\vec{x}_k - \vec{c}_i\|_2. \tag{17}$$

Normally, $n_s \geq n_c$. When $n_s = n_c$, it is the interpolation model. Each training data is chosen as the center of the RBF, and the weights can be calculated as

$$\vec{\omega}^* = \boldsymbol{K}^{-1}\vec{y}, \tag{18}$$

where $\vec{y} = \{y(\vec{x}_1), y(\vec{x}_2), \ldots, y(\vec{x}_{n_s})\}^T \in \mathbb{R}^{n_s}$ is the output vector and the ij-th element of matrix $\boldsymbol{K}$ is calculated as $\|\vec{x}_i - \vec{x}_j\|_2$. When $n_s > n_c$, it becomes the regression model. The centers can be selected from the training data via a k-means algorithm [48]. The weights can be computed using the least square method as follows:

$$\vec{\omega}^* = (\boldsymbol{K}^T\boldsymbol{K})^{-1}\boldsymbol{K}^T\vec{y}, \tag{19}$$

where the ij-th element of matrix $\boldsymbol{K}$ is calculated as $\|\vec{x}_i - \vec{c}_j\|_2$.

In the l-th environment, when estimating an individual's past fitness value in the $(l-l_0)$-th environment and at its position $\vec{x}$, we first check whether the tuple $(\vec{x}, l-l_0, f(\vec{x}, \vec{\alpha}_{l-l_0}))$ exists in the database. If yes, we can use the recorded fitness value. Otherwise, we construct a local RBF model using n_s nearest neighbors of $\vec{x}$ in the $(l - l_0)$-th environment, and obtain the estimated value $\hat{f}(\vec{x}, \vec{\alpha}_{l-l_0})$. Euclidean distance is used here. Make sure that a data sample should not be stored in the database more than once to avoid singularity of the matrix $\boldsymbol{K}$ in constructing an RBF model.

To get good approximation models, the training data should be properly distributed in the search space. To this end, we employ the symmetric Latin hypercube design (SLHD) [49] to generate the initial population, and once the environment changes, we generate half of the population size individuals using SLHD to replace the worst half of the population. SLHD has the same characteristics as the regular Latin hypercube design (LHD). To avoid ill-distributed training data, we use the approach described in [25] to detect severe outliers and replace them with an estimated fitness value averaged over the neighboring points in the decision space in that time instant.

5.2 Autoregressive model

An AR model of order ψ is denoted by AR (ψ). Given a time series of data X_l, a typical AR (ψ) model takes the form [15]

$$X_l = \epsilon_l + \sum_{i=1}^{\psi} \eta_i X_{l-i}, \tag{20}$$

where ϵ_l is white noise, $\vec{\eta} = \{\eta_1, \eta_2, \ldots, \eta_\psi\}^T$ denotes the vector of the parameters of the model. To perform the prediction task, we form the input-output pair as $(x_l, y_l) = (\{X_{l-1}, \ldots, X_{l-\psi}\}^T, X_l)$. Hence in order to predict y_l, the training data are $(x_{l-1}, y_{l-1}), \ldots, (x_{l-n_s}, y_{l-n_s})$ [50]. Once the value of ψ is determined, the parameters of the model can be estimated using the least square method. Specifically, the coefficient can be calculated as

$$\vec{\eta}^* = (\vec{\chi}^T \vec{\chi})^{-1} \chi^T \vec{Y}, \tag{21}$$

where $\vec{\chi} = \{x_{l-1}, \ldots, x_{l-n_s}\}^T$ and $\vec{Y} = \{y_{l-1}, \ldots, y_{l-n_s}\}$. For regression, we ensure that $n_s > \psi$. The training data are obtained using the RBF approximation model described in the previous section.

The aforementioned RBF model can also be used to perform the prediction task. Similar to the RBF approximator, in order to predict the value at time l, i.e., y_l, we retrieve n_s nearest historical data of x_l from the database. The distance is also measured in Euclidean distance but we apply it in two different spaces. One is to retrieve x_l's neighbors in x_l's space, the other is in x_l^*'s space, where $x_l^* = x_l - x_{l-1} = \{y_{l-1} - y_{l-2}, y_{l-2} - y_{l-3}, \ldots, y_{l-\psi} - y_{l-\psi-1}\}^T$ [51]. The former focuses on ψ-length historical series with similar values, while the latter pays more attention to the historical series with similar dynamics. We denote the former RBF model by RBF(ψ) and the latter by GRBF(ψ).

5.3 PSO as the optimizer

We adopt three variants of the particle swarm optimization (PSO) algorithm [52,53] as the optimizer, including a PSO with a simple re-start strategy (denoted by rPSO), a PSO with a memory scheme used in [6] (denoted by memPSO), and a PSO using species technique [54,55] (denoted by SPSO). memPSO and SPSO represent two typical TMO techniques to DOPs, one using memory scheme and niching [56] and the other multiple populations.

In the following, we provide a brief description of rPSO. In a swarm of size μ, the i-th particle's position is denoted by $\vec{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in})$, the best position it has found so far is denoted as $\vec{p}_i = (p_{i1}, p_{i2}, \ldots, p_{in})$ (called *pbest*), the best position of the whole population (called *gbest*) or of the current particle's neighborhood (called *lbest*) is denoted by $\vec{p}_b = (p_{b1}, p_{b2}, \ldots, p_{bn})$, and the rate to change the position of this particle is called velocity and is denoted by $\vec{v}_i = (v_{i1}, v_{i2}, \ldots, v_{in})$, where $i = 1, 2, \ldots, \mu$.

The basic PSO used in this work is the PSO using the constriction factor [57], in which, at each iteration step g, the i-th particle updates its d-th dimension of velocity according to Eq. 22 and position in the search space according to Eq. 23 as follows:

$$v_{id}(g) = \chi(v_{id}(g-1) + c_1 r_1(p_{id} - x_{id}(g-1)) + c_2 r_2(p_{bd} - x_{id}(g-1))) \tag{22}$$

$$x_{id}(g) = x_{id}(g-1) + v_{id}(g), \tag{23}$$

where

$$\chi = \frac{2}{|2 - c - \sqrt{c^2 - 4c}|} \quad c = c_1 + c_2 \quad c > 4.0. \tag{24}$$

The constriction factor χ provides a damping effect on a particle's velocity and ensures that the particle will converge over time. c_1 and c_2 are constants, typically 2.05, and thus, $\chi = 0.729844$ according to Eq. 24. r_1 and r_2 are random numbers uniformly distributed in [0, 1]. Moreover, the velocity $\vec{v}_i$ can be constricted within the range of $[-V_{MAX}, +V_{MAX}]$. In this way, the likelihood of a particle's flying out of the search space is reduced. The value of $\pm V_{MAX}$ is usually set to be the lower and upper bounds of the allowed search ranges as suggested in [53].

We refer readers to [6,54,58] for a detailed description of memPSO and SPOS.

5.4 Computational complexity

We first briefly present an analysis of the computational costs for the construction of an RBF model and an AR model, and then analyze the overall computational overhead of the algorithm $(p+q)$-POA over a standard POA.

(1) **Compute the Euclidean distance** between the model fitting point and all records with the same environment index (e.g., the index l) in the database. This costs time in the order $O(n_{db_l}d)$, where n_{db_l} is the number of the records in the database in the l-th environment and d is the dimensionality of the search space.
(2) **Sort the above recorded data** with regard to their Euclidean distances to the fitting point. The time complexity of using the *Quicksort* algorithm [59] is $O(n_{db_l}log_2 n_{db_l})$ on average and $O(n^2_{db_l})$ in the worst case.
(3) **Computing coefficients of the RBF and AR models** Since we ensure that no singular matrix exists in the calculation, for interpolation, we employ LU decomposition [60] to solve Eq. 18, and for regression, we employ Cholesky decomposition [61] to solve Eqs. 19 and 21. Both methods have a complexity of $O(n_r^3)$, where n_r is the rank of the matrix involved in the above calculation. Specifically, $n_r = n_c$ for the RBF model and $n_r = \psi$ for the AR model.

Therefore, the overall complexity for the construction of an RBF and an AR models is

$$O(n_{db_l}d + n^2_{db_l} + n_c^3 + \psi^3). \tag{25}$$

For the $(p+q)$-POA, to estimate p past values, p RBF approximators are needed while for the q future values only one AR predictor is enough. Therefore, if the database size n_{db} is the same in each environment, the overall computational overhead over a fitness evaluation is

$$O(pn_{db}d + pn^2_{db} + pn_c^3 + \psi^3). \tag{26}$$

Normally, n_{db} is much larger than d, n_c, and ψ. Thus the computational overhead is mainly determined by

$$O(pn^2_{db}). \tag{27}$$

6 Test problems

As ROOT can be regarded as a more practical way of addressing DOPs, we can create test problems for ROOT via modifying existing benchmark problems for DOPs. The first test problem is the modified moving peak benchmark (mMPB) [9]. It was derived from MPB benchmark [33] by allowing each peak to have its own change severities of width and height. In this way, some parts of the search space change more severely than other parts, which is very useful in evaluating solutions that are robust over time. Based on a similar idea, we propose further in this study the modified dynamic rotation peak benchmark generator (mDRPBG) and the modified dynamic composition benchmark generator (mDCBG). mDRPBG comes from DRPBG [62] by allowing each peak to have its own width change severity and height change severity, while mDCBG is obtained from DCBG [62] by allowing the height of each composition part to have its own change severity. The three benchmark problems are described as follows.

6.1 TP1-modified moving peak benchmark

We use the cone peak function and do not consider any base landscape in mMPB. An n-dimensional test function with m peaks is defined as:

$$F(\vec{x}, t) = \max_{i=1}^{m}\{H_i(t) - W_i(t) \cdot \|\vec{x} - \vec{X}_i(t)\|_2\}, \tag{28}$$

where $\vec{H}$, $\vec{W}$, $\vec{X}$ denote the peak height, width and position, respectively.

For every Δe evaluations the height and width of the i-th peak are changed as follows:

$$\begin{aligned} H_i(t+1) &= H_i(t) + \text{height_severity}_i \cdot N(0,1) \\ W_i(t+1) &= W_i(t) + \text{width_severity}_i \cdot N(0,1), \end{aligned} \tag{29}$$

where $N(0,1)$ denotes a normally distributed one-dimensional random number with a mean of zero and variance one, $\vec{\text{height_severity}}$ and $\vec{\text{width_severity}}$ denotes the height change severity and width change severity respectively. The position is moved by a vector $\vec{v}_i$ of a fixed length s in a random direction ($\lambda = 0$) or a direction exhibiting a trend ($\lambda > 0$) as follows:

$$\vec{X}_i(t+1) = \vec{X}_i(t) + \vec{v}_i(t+1). \tag{30}$$

The shift vector $\vec{v}_i(t+1)$ is a linear combination of a random vector $\vec{r}$ and the previous shift vector $\vec{v}_i(t)$, and is normalized to a length of s, i.e.

$$\vec{v}_i(t+1) = \frac{s}{\|\vec{r} + \vec{v}_i(t)\|_2}((1-\lambda)\vec{r} + \lambda\vec{v}_i(t)). \tag{31}$$

The random vector $\vec{r}$ is created by drawing random numbers in $[-0.5, 0.5]$ for each dimension and normalizing its length to s.

6.2 TP2-modified dynamic rotation peak benchmark generator

We use the random change type in mDRPBG, so the differences between mMPB and mDRPBG are the definition of

the peak function and the way of changing the peak position. An n-dimensional test function with m peaks is defined as:

$$F(\vec{x},t)=\max_{i=1}^{m}\{H_i(t)/(1+W_i(t)\cdot\|\vec{x}-\vec{X}_i(t)\|_2/\sqrt{n})\} \quad (32)$$

where $\vec{H}$, $\vec{W}$, $\vec{X}$ denote the peak height, width and position, respectively.

For every Δe evaluations the height and width of the i-th peak change in the same way as mMPB in Eq. 29. The i-th peak position $\vec{X}_i$ is changed by rotating its projection on randomly paired dimension from the first dimension axis to the second by an angle θ_i. The dynamics of the rotation angle $\vec{\theta}$ and the specific rotation algorithm [62] are shown below:

Step 1. $\theta_i(t+1)=\theta_i(t)+\theta_\text{severity}\cdot N(0,1)$.
Step 2. Randomly select l dimensions (l is an even number) from the n dimensions to compose a vector $r=[r_1,r_2,\ldots,r_l]$.
Step 3. For each pair of dimension $r[j]$ and dimension $r[j+1]$, construct a rotation matrix $R_{r[j],r[j+1]}(\theta_i(t+1))$.
Step 4. A transformation matrix for the i-th peak $A_i(t+1)$ is obtained by:
$A_i(t+1)=R_{r[1],r[2]}(\theta_i(t+1))\cdot R_{r[3],r[4]}(\theta_i(t+1))\cdots R_{r[l-1],r[l]}(\theta_i(t+1))$
Step 5. $\vec{X}_i(t+1)=\vec{X}_i(t)\cdot A_i(t+1)$,

where the rotation matrix [63] $R_{r[j],r[j+1]}(\theta_i(t+1))$ is obtained by rotating the projection of $\vec{X}_i(t)$ on the plane $r[j]-r[j+1]$ by an angle $\theta_i(t+1)$ from the $r[j]$-th axis to the $r[j+1]$-th axis. As for the value of l, if n is an even number, $l=n$; otherwise $l=n-1$.

6.3 TP3-modified dynamic composition benchmark generator

We use base functions with the original optimum being $\vec{0}$, so the composition function with m base functions and n dimensions can be described as:

$$F(\vec{x},t)=\sum_{i=1}^{m}\left(w_i\cdot\left(f_i'\left(\frac{\vec{x}-\vec{O}_i(t)}{\lambda_i}\cdot\vec{M}_i\right)+H_i(t)\right)\right), \quad (33)$$

where $f_i(\vec{x})$ is the i-th base function, $\vec{M}_i$ is the orthogonal rotation matrix for each $f_i(\vec{x})$, and $\vec{O}_i(t)$ is the optimum of the changed $f_i(\vec{x})$ caused by rotating the landscape at the time t. The weight value w_i for each $f_i(\vec{x})$ is calculated as:

$$w_i=e^{-\sqrt{\frac{\sum_{k=1}^{n}(x_k-o_i^k)^2}{2n\sigma_i^2}}},$$

$$w_i=\begin{cases} w_i & \text{if } w_i=max(w_i)\\ w_i\cdot(1-max(w_i)^{10}) & \text{if } w_i\neq max(w_i),\end{cases}$$

$$w_i=w_i/\sum_{i=1}^{m}w_i,$$

where for each base function $f_i(\vec{x})$, σ_i is the convergence range factor and λ_i is the stretch factor, which is defined as:

$$\lambda_i=\sigma_i\cdot\frac{X_{max}-X_{min}}{x_{max}^i-x_{min}^i},$$

where $[X_{max},X_{min}]^n$ is the search range of $F(\vec{x})$ and $[x_{max}^i,x_{min}^i]^n$ is the search range of $f_i(\vec{x})$. In Eq. 33, $f_i'(\vec{x})=C\cdot f_i(\vec{x})/|f_{max}^i|$, where C is a predefined constant and f_{max}^i is the estimated maximum value of $f_i(\vec{x})$, which is estimated as:

$$f_{max}^i=f_i(\vec{x_{max}}\cdot\vec{M}_i)$$

In the mDCBG, $\vec{M}$ is initialized using the same transformation matrix construction algorithm as the one in the mDRPBG and then remains unchanged. For every Δe evaluations, $\vec{H}$ and $\vec{O}$ change in the same way as the parameters $\vec{H}$ and $\vec{X}$ in mDRPBG. In this study, we only select the Sphere's function as the base function which takes the form

$$f(\vec{x})=\sum_{i=1}^{n}x_i^2 \quad x_i\in[-100,100]. \quad (34)$$

7 Empirical results

In this section, the first goal is to investigate the performance of the POA designed for TMO in the context of ROOT. We want to see if they can be easily adapted to ROOT, and if so, what a performance they can achieve. Second, we would like to evaluate the effectiveness of the proposed framework for ROOT and compare it to that of the existing TMO EAs. Finally, we empirically analyze the influence of each component in the framework on the performance so that we can provide suggestions on how to select proper approximators and predictors. Before presenting the empirical results, we provide the parameter setting and the performance indicators used in the experiments.

7.1 Parameter setting

We have a built-in random number generator for each test problem, and to generate test instances we always set the generator seed to 1. For each problem, we generate instances of a dimension 2, 5, and 10, and thus we have 9 test instances in total. The parameters height_severity and width_severity are initialized by drawing random numbers within $[1,10]^m$ and $[0.1,1]^m$, respectively, and then fixed. The change frequency is measured in function evaluations (FEs). All investigated algorithms were run 25 times, each with different initial populations in each instance. The parameter settings of test problems are summarized in Table 1.

All parameters of the PSOs studied in this work are listed in Table 2. To examine the influence of the time interval

Table 1 Parameters setting of test problems

Parameters	TP1	TP2	TP3
Number of changes	50	50	50
Δe(FEs)	2,500	5,000	10,000
m	5	5	5
n	2, 5, 10	2, 5, 10	2, 5, 10
Search range	$[0, 50]^n$	$[-5, 5]^n$	$[-5, 5]^n$
Height range	[30, 70]	[10, 100]	[10, 100]
Initial height	50	50	50
Width range	[1, 12]	[1, 10]	N/A
Initial width	6	5	N/A
Rotation angle range	N/A	$[-\frac{\pi}{6}, \frac{\pi}{6}]$	$[-\frac{\pi}{6}, \frac{\pi}{6}]$
Initial angle	N/A	0	0
Height_severity range	[1, 10]	[1, 10]	[1, 10]
Width_severity range	[0.1, 1]	[0.1, 1]	N/A
θ_Severity	N/A	0.1	0.1
Other parameters	$\lambda = 0$	N/A	$C = 2{,}000$
	$s = 1.0$		$\sigma_i = 1.0, i = 1, 2, \ldots m$
δ_{drop}	0.2	0.2	0.2

Table 2 Parameters setting of PSOs

Parameters	TP1	TP2	TP3
PSO			
$c_1 = c_2 = 2.05, \chi = 0.729844$			
Swarm size: $\mu = 50$			
V_{MAX}	50	10	10
SPSO			
r_s	15	3	3
P_{MAX}	10	10	10
memPSO			
Premature convergence threshold: 0.01			
Loss of diversity threshold: 0.001			
r_s	15	3	3
P_{MAX}	10	10	10
δ_{pre}	15	30	50

$p + q$ on the performance of the framework, for $(p + q)$-PSO, we vary the value $p+q$ from 1 to 10 and for each $p+q$ value, we vary the percentage of p from 0 to 100 %. For the RBF used in the instantiation, we set $n_c = (D+1)(D+2)/2$ [14], where D is the dimension of the input vector. When regression is used, the number of training data is set to be more than twice of the model size (i.e., n_c of RBF and ψ of AR). Note that for each investigated algorithm on each test problem, we assume that the environment changes can be detected [64,65]. Consequently, we focus on the algorithms' capability of addressing ROOT problems.

7.2 Performance evaluation

Let $S = \langle S_1, S_2, \ldots, S_k \rangle$ be a sequence of solutions found by an algorithm for the problem defined in Eq. 7 ($\langle f(\vec{x}, \vec{\alpha}_1), f(\vec{x}, \vec{\alpha}_2), \ldots, f(\vec{x}, \vec{\alpha}_l) \rangle$). A solution is regarded as robust if it is used for at least two consecutive problem instances and thus we have $1 \leq k \leq l$. We first define a measure error ϵ_i and a measure sensitivity D_i for a single solution S_i as follows:

$$\epsilon_i = \frac{1}{N_i} \sum_{j=l_0}^{l_0+N_i-1} |\text{opt}_j - f(S_i, \vec{\alpha}_j)|, \tag{35}$$

$$D_i = \sqrt{\frac{1}{N_i - 1} \sum_{j=l_0}^{l_0+N_i-1} (|\text{opt}_j - f(S_i, \vec{\alpha}_j)| - \epsilon_i)^2}. \tag{36}$$

In Eqs. 35 and 36, N_i is the number of different environments in which S_i is used, $f(\vec{x}, \vec{\alpha}_{l_0})$ is the first environment S_i is used, $f(S_i, \vec{\alpha}_j)$ is the fitness value of the solution S_i, and opt_j is the fitness value of the true optimum of the j-th problem instance. Note that in Eq. 36, if the denominator is N_i, for solutions that are not robust, D_i will be zero but these solutions are not good in the sense of ROOT. Therefore, we set the denominator to be $N_i - 1$ to avoid this situation. This also means that D_i is significant only for robust solutions over time.

Now we denote the sequence of all robust solutions by $S_r = \langle S_{r_1}, S_{r_2}, \ldots, S_{r_k} \rangle$ $(1 \leq r_k \leq k)$, and we denote the index sequence $\langle r_1, r_2, \ldots, r_k \rangle$ by R. With the two measures above, we can resort to the following three measures to evaluate a solution sequence found by an algorithm.

- ρ – the robustness rate of the solution sequence:

$$\rho = 1 - \frac{k-1}{l-1}, \tag{37}$$

  where we add -1 to the numerator and the denominator because we need at least one solution for a problem and we have $\rho \in [0, 1]$. When $k = 1$, the ideal situation happens and $\rho = 1$. When $k = l$, no robust solutions have been found and $\rho = 0$.
- E_{avg} – the average error of the solution sequence:

$$E_{\text{avg}} = \frac{1}{k} \sum_{j=1}^{k} \epsilon_j. \tag{38}$$

  We can see that E_{avg} measures how close the sequence of solutions is to the true optima over time.
- D_{avg} – the average sensitivity of the robust solution sequence:

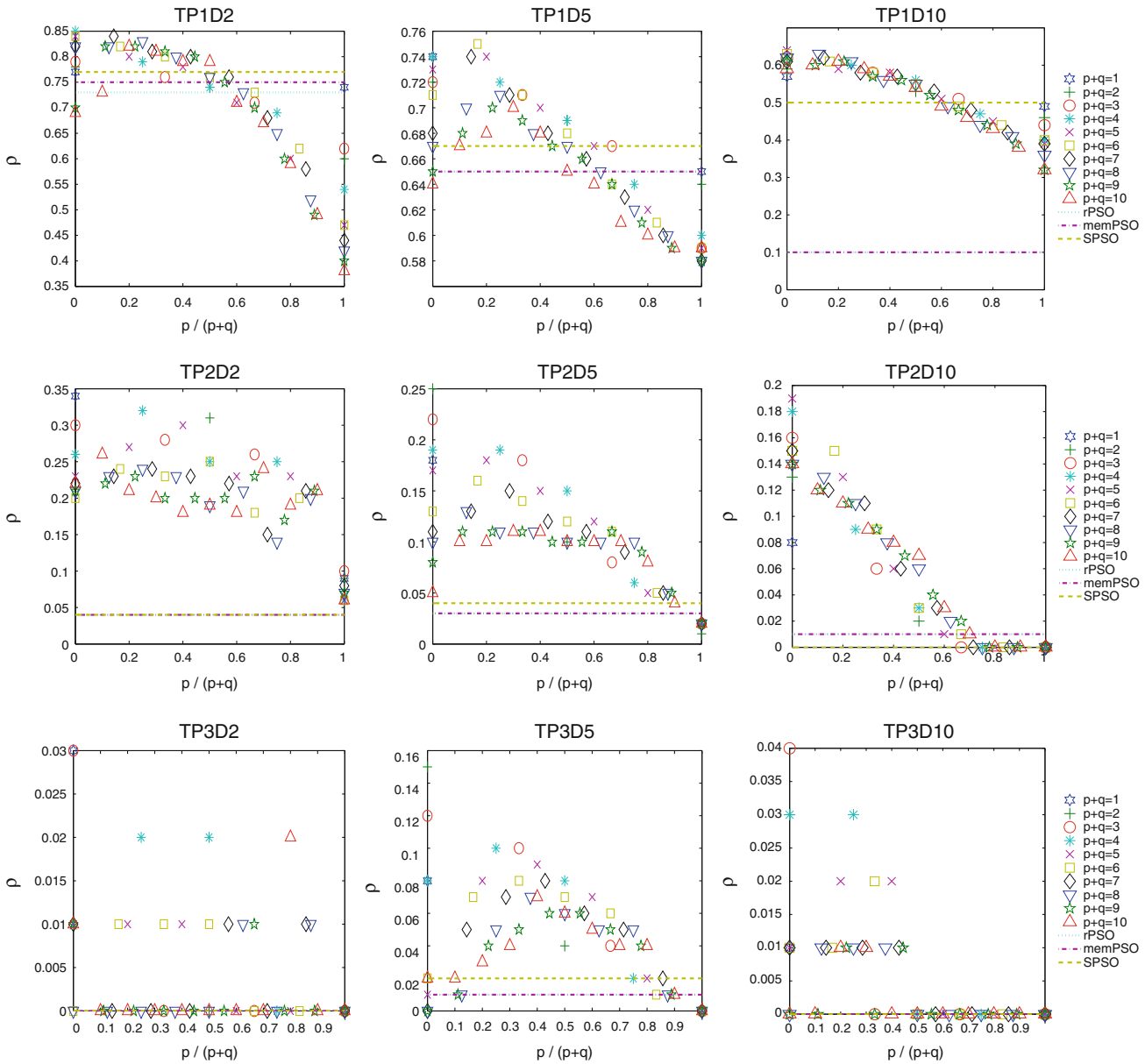

Fig. 2 The robustness rate ρ of the solution sequence obtained via varying the $(p+q)$ configuration and the length of optimization interval

$$D_{\text{avg}} = \frac{1}{|R|} \sum_{j \in R} D_j, \tag{39}$$

where $|R|$ is the length of the sequence R, which equals k, D_{avg} measures the general sensitivity of the sequence of robust solutions to the environmental changes.

From the above definitions, it can be seen that the goal of TMO is to minimize E_{avg}, where $\rho = 0$ and D_{avg} do not exist. In contrast, the primary goal of the ROOT approach to DOPs pursued in this paper is to maximize ρ, i.e., to find as many robust solutions over time as possible.

In addition, we can also consider the success rate ξ of running an algorithm for several times on a dynamic problem. A run is successful if at least one robust solution is found by the algorithm.

The final solution sequence is obtained as follows: The first solution for the first environment is the best solution found so far by the algorithm. For the j-th environment where $2 \le j \le l$, if the solution in the $(j-1)$-th environment is $S_i (1 \le i \le j-1)$ and if

$$\left| \frac{f(S_i, \vec{\alpha}_j) - opt_j}{opt_j} \right| \le \delta_{\text{drop}}, \tag{40}$$

then solution S_i will still be used for the j-th environment. Otherwise, the best solution found so far by the algorithm for the j-th environment will be adopted as the solution S_{i+1}. The parameter δ_{drop} denotes the maximal tolerance of

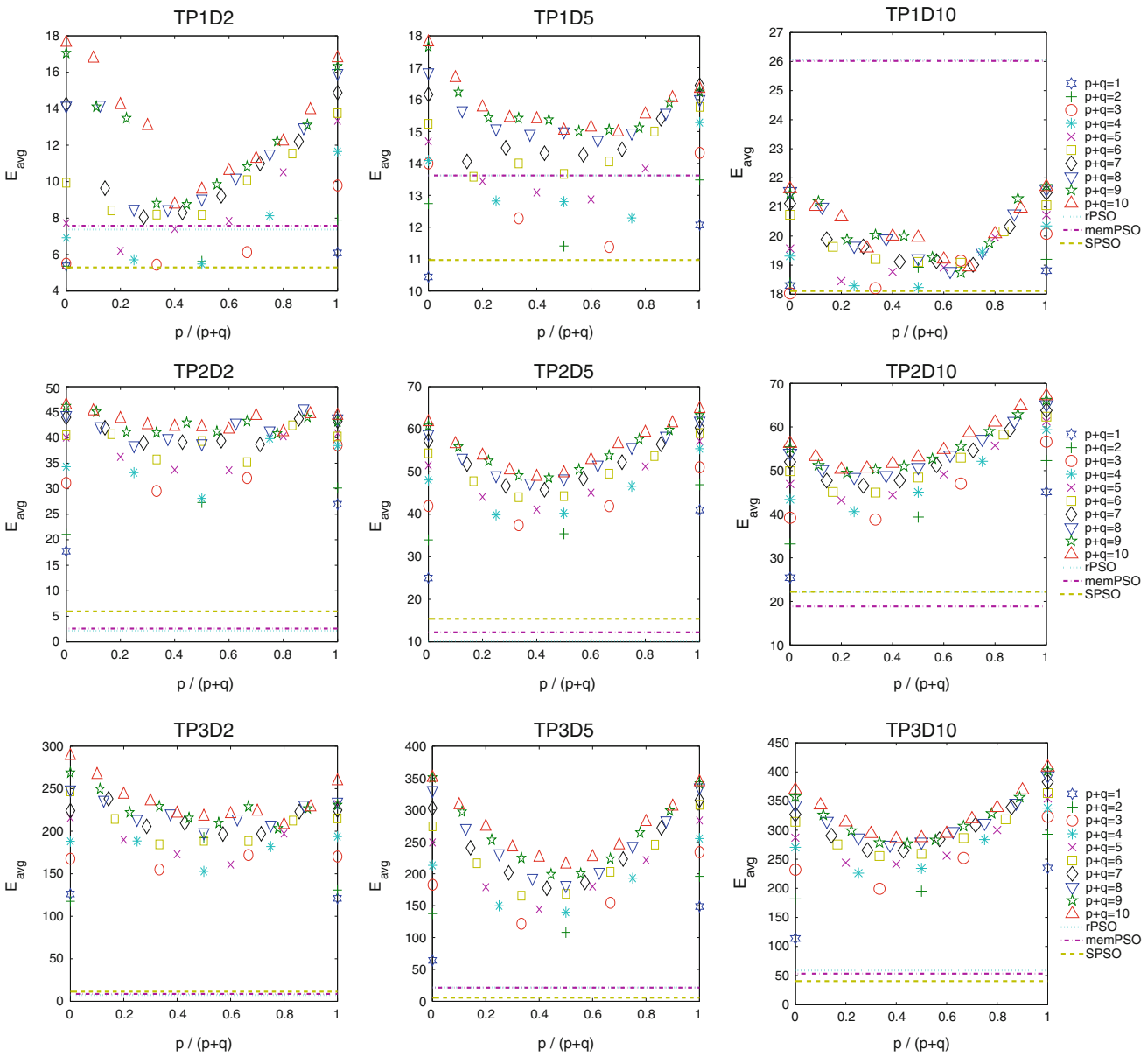

Fig. 3 The average error of the solution sequence obtained via varying the $(p+q)$ configuration and the length of optimization interval

performance degradation of the solutions when the environment changes.

7.3 Simulation results

To examine the effectiveness of the framework, $(p+q)$-PSO works on the true past and future fitness values. The experimental results regarding ρ, E_{avg} and D_{avg} of the investigated algorithms on test problems were plotted in Figs. 2, 3 and 4. In Fig. 4, the missing data points and lines indicate that the corresponding algorithms failed to find robust solutions over time. The above results were also summarized in Table 3, where the resultant success rates of finding robust solutions over time (i.e., ξ) were included and (1+4)-PSO was selected as a representative. In Table 3, the values in bold are at a 0.05 significance level using the Wilcoxon rank sum test. To take a closer look at the behaviors of the algorithms, Fig. 5 shows the evolutionary curves with respect to the performance error and the Euclidean distance in the search space between the best solution found so far and the global optimum for the test problems with a dimension of 5. The plots were based on the data collected for the run with the median performance regarding ρ among 25 runs.

Results in Table 3 clearly suggest that $(1+4)$-PSO succeeded in finding robust solutions over time in all instances of test problems and outperformed traditional TMO algorithms

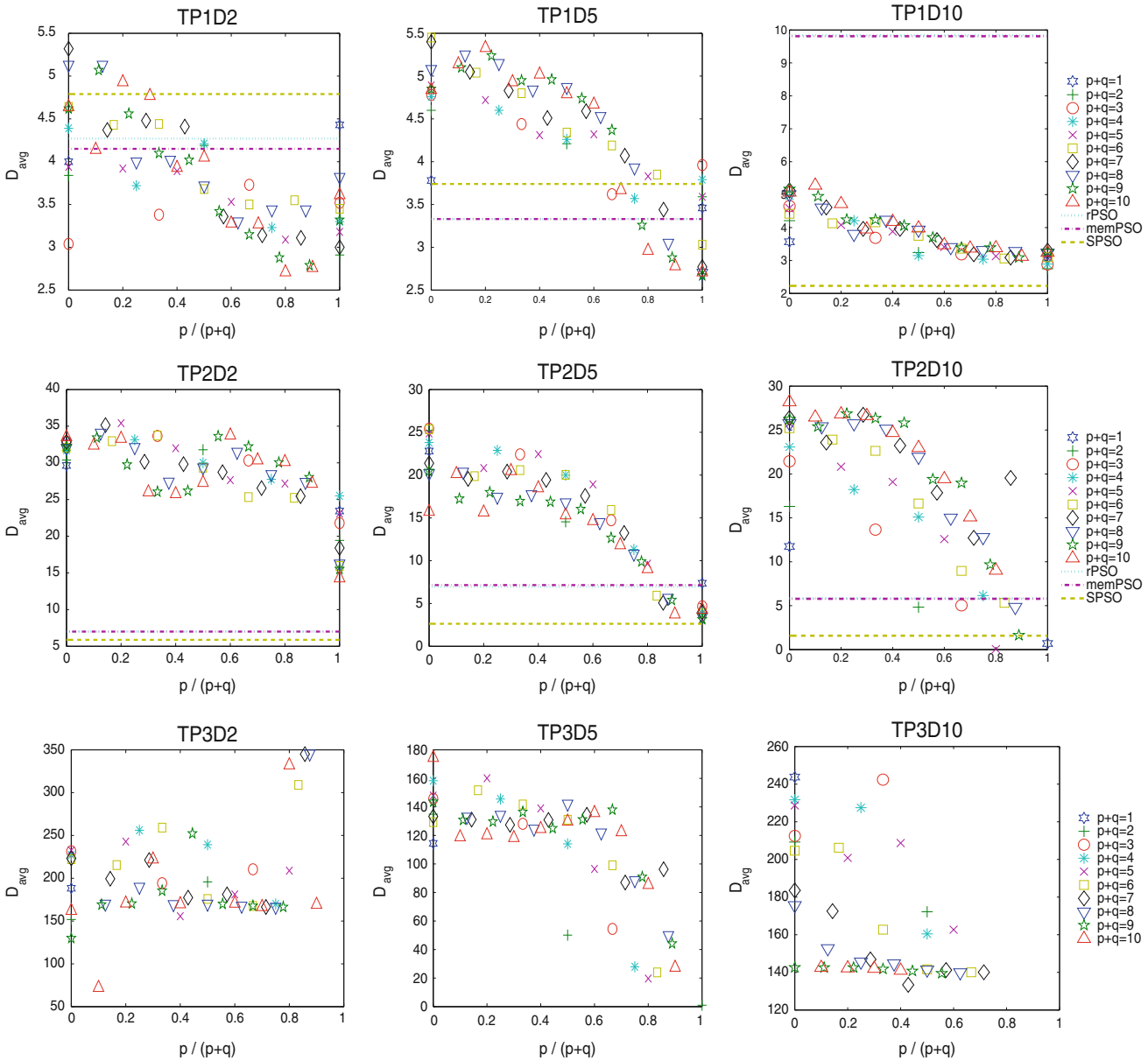

Fig. 4 The average standard deviation of the solution sequence obtained via varying the $(p+q)$ configuration and the length of optimization interval

with respect to the success rate and the robustness rate. This is expected because (1+4)-PSO explicitly takes into account the robustness in the time domain. This can also be verified by Fig. 2 suggesting that most $(p+q)$-PSO algorithms achieved better robustness rates than the TMO algorithms.

7.3.1 Traditional TMO algorithms for DOPs

As can be seen in Table 3, the TMO algorithms succeeded in finding robust solutions over time on 5-dimensional problems TP1, TP2, and TP3. This confirms the expectation that some techniques, such as reusing historical information and maintaining the diversity for TMO, are beneficial to ROOT as well. Figure 2 clearly shows that SPSO obtained the best performance among the three algorithms. This indicates that diversity plays an important role in both TMO and ROOT, because by maintaining a certain level of diversity, the optimizer can keep its search ability. The performance of memPSO is similar to that of rPSO, since the historical information stored in the memory was not used for finding robust solutions over time but tracking the global optima.

Additionally, it can be observed from Fig. 3 that the TMO algorithms achieved better E_{avg} results than most of the $(p+q)$-PSO algorithms. This is natural since minimizing

Table 3 Overall performance of the investigated algorithms

Test problems	Investigated algorithms	Dimension 2				Dimension 5				Dimension 10			
		ξ	ρ	E_{avg}	D_{avg}	ξ	ρ	E_{avg}	D_{avg}	ξ	ρ	E_{avg}	D_{avg}
TP1	rPSO	1.00	0.73 (0.04)	7.37 (1.12)	4.27 (0.48)	1.00	0.65 (0.00)	13.62 (0.06)	**3.33** (0.03)	1.00	0.10 (0.09)	26.06 (2.65)	9.84 (1.44)
	memPSO	1.00	0.75 (0.03)	7.58 (1.11)	4.15 (0.44)	1.00	0.65 (0.00)	13.62 (0.06)	**3.33** (0.02)	1.00	0.10 (0.09)	26.02 (2.66)	9.81 (1.43)
	SPSO	1.00	0.77 (0.00)	**5.29** (0.12)	4.79 (0.08)	1.00	0.67 (0.04)	**10.97** (2.11)	3.74 (0.55)	1.00	0.50 (0.04)	**18.11** (1.97)	**2.23** (0.19)
	(1+4)-PSO	1.00	**0.80** (0.01)	6.20 (0.11)	**3.92** (0.16)	1.00	**0.74** (0.05)	13.44 (2.76)	4.72 (0.78)	1.00	**0.59** (0.04)	18.45 (1.86)	4.08 (0.84)
TP2	rPSO	1.00	0.04 (0.00)	**2.17** (0.94)	7.00 (0.02)	0.92	0.03 (0.02)	**10.07** (7.12)	6.95 (2.33)	0.32	0.01 (0.01)	22.16 (8.91)	5.84 (0.76)
	memPSO	0.96	0.04 (0.01)	2.58 (1.48)	7.00 (0.02)	0.92	0.03 (0.02)	12.23 (5.85)	7.15 (2.24)	0.40	0.01 (0.01)	**18.85** (3.88)	5.78 (0.58)
	SPSO	1.00	0.04 (0.00)	5.96 (0.41)	**5.88** (1.01)	0.92	0.04 (0.02)	15.41 (1.18)	**2.63** (1.55)	0.20	0.00 (0.01)	22.21 (1.50)	**1.58** (2.13)
	(1+4)-PSO	1.00	**0.27** (0.02)	36.23 (1.66)	35.41 (1.47)	1.00	**0.18** (0.03)	44.06 (1.56)	20.82 (4.98)	1.00	**0.13** (0.03)	43.17 (1.35)	20.82 (3.47)
TP3	rPSO	0.00	0.00 (0.00)	**7.35** (2.31)	$\sim$	0.48	0.01 (0.01)	21.74 (10.28)	**1.45** (0.62)	0.00	0.00 (0.00)	58.79 (12.52)	$\sim$
	memPSO	0.00	0.00 (0.00)	8.75(1.55)	$\sim$	0.68	0.01 (0.01)	21.41 (7.03)	1.72 (0.82)	0.00	0.00 (0.00)	53.31 (9.53)	$\sim$
	SPSO	0.00	0.00 (0.00)	11.41 (0.81)	$\sim$	0.88	0.02 (0.01)	**5.68** (0.91)	2.87 (0.91)	0.00	0.00 (0.00)	**40.51** (4.80)	$\sim$
	(1+4)-PSO	0.28	**0.01** (0.02)	190.00 (5.44)	242.80 (42.66)	1.00	**0.08** (0.02)	178.89 (10.64)	160.08 (16.41)	0.56	**0.02** (0.02)	244.05 (13.60)	200.83 (16.09)

the average error between the obtained solutions and the true optima is consistent with the goal of TMO. According to Fig. 4, the TMO algorithms showed better D_{avg} performances than most of the $(p+q)$-PSO algorithms on most TP1 and TP2 test problems. The reason is that although these algorithms' goal is to track the moving optima, the optima change only slightly during some time intervals. As a result, these optima found by the TMO algorithms are also the robust solutions over time. Evidence supporting the above reasoning can be found in Fig. 5. For example on the problem "TP1D5", from the 500th to the 1000th generation, the optima changed very slowly so the optimal solution found at the 500-th generation could be used during this time interval with acceptable performance.

To summarize, the TMO algorithms without explicitly considering the robustness in the time domain can find robust solutions over time only if the global moving optimum happens to be a robust optimal solution over time.

7.3.2 The $(p+q)$-PSO framework

In this section, we mainly investigate the influence of configuration (i.e., the values of p and q) of $(p+q)$-PSO on its performance. From the results plotted in Figs. 2, 3, 4 and 5, we can make some observations by comparing the framework's performances with different configurations.

First, the robustness rate does not always increase as the length of the time interval increases. Generally, there is an optimal length of the time interval for a given problem. As can be seen from Fig. 2, the optimal length ranges from 4 to 8 on TP1, from 1 to 5 on TP2, and from 1 to 3 on TP3. The optimal value of $p+q$ is problem-dependant. According to Fig. 5, TP2 and TP3 change much more severely than TP1, so the likelihood of using a solution for a long time interval on TP2 and TP3 was smaller than that on TP1. This is why larger values of $p+q$ favored TP1 while smaller values are more appropriate for TP2 and TP3.

Second, the following observations can be made regarding the ratio of p to the length of $p+q$. As Fig. 2 illustrates, for 2-dimensional and 5-dimensional TP1, TP2 and TP3D5, increasing p first improved the performance. However, further increasing p was detrimental. According to Fig. 2, for TP1, the performance was even worse than that of the TMO when $p > q$. For TP1D10, TP2D10, and TP3 of dimensions 2 and 10, increasing p generally degraded the performance. The above phenomena can be explained as follows. If there are some "consistencies" in the dynamics during some periods of time, past values are beneficial, and when p is larger than the length of the period, the outdated information will mislead the search and hence degrades the performance. The "consistency" in the dynamics here means that successive environments share some common characteristics or in other words the environment does not change severely.

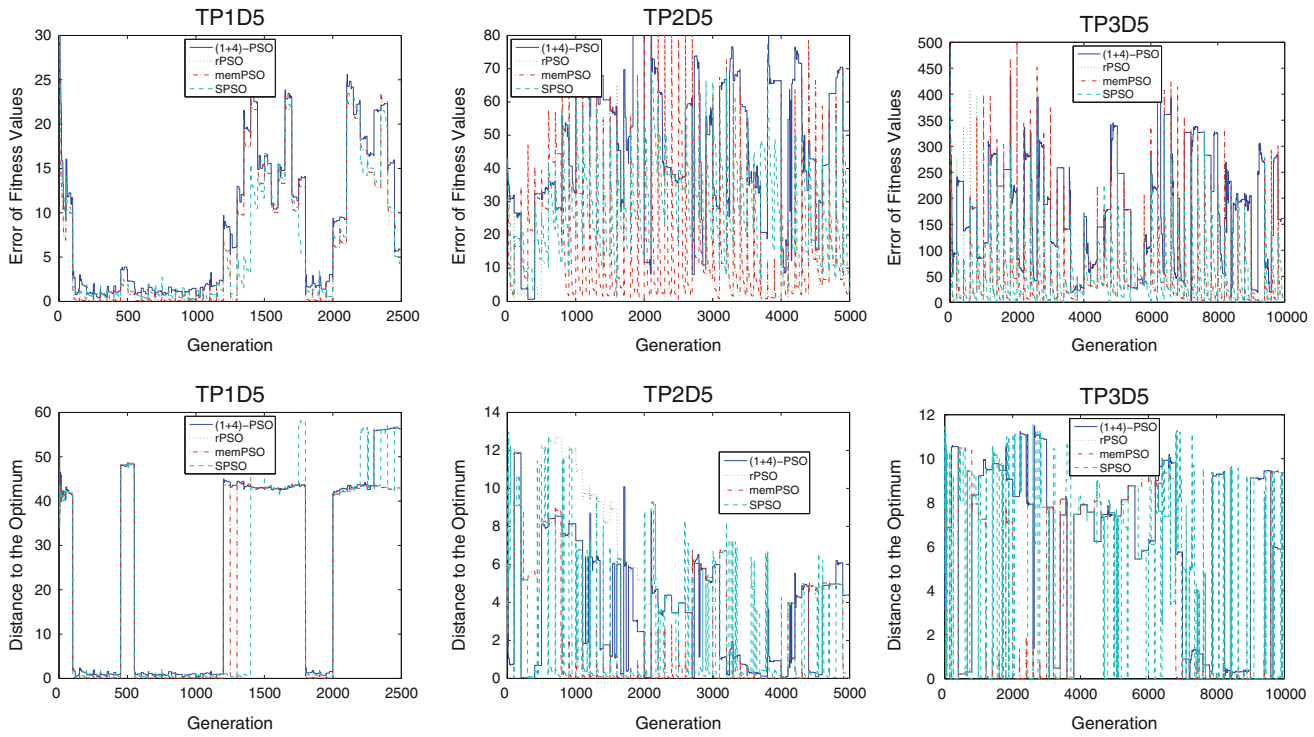

Fig. 5 The evolutionary curves with respect to the performance error and the Euclidean distance in the search space between the best solution found so far and the global optimum. The plots show the median performance with respect to the robustness rate ρ among 25 runs for a dimension of 5

Taking TP3D5 as an example, seen from Fig. 5, there were some periods such as the first 1000 generations and the period around the 4000-th generation the degrees of the changes were mild. This is why on TP3D5 there existed some "sweet spots" for the value of p.

Third, it can be seen from Figs. 3 and 4 that some algorithms perform better regarding E_{avg}, but worse regarding D_{avg}. These results verify the analyses in Sect. 3.2 that in some situations E_{avg} and D_{avg} are two conflicting goals. In fact, from Figs. 2, 3 and 4, we find that ρ, E_{avg} and D_{avg} cannot be simultaneously optimized. For instance, when $\rho = 0$, there is no robust solution over time and ROOT degenerates to TMO. Consequently, the best E_{avg} performance is obtained. When $\rho = 1$, only one solution is used for all environments so the worst D_{avg} performance is achieved. To illustrate this, we have plotted all the available results of $(p + q)$-PSO in the $E_{\text{avg}} - D_{\text{avg}} - \rho$ space in Fig. 1. From this figure, we can clearly see that there existed some points, among which one point could not outperform others regarding all the three measures simultaneously.

8 Conclusions

Different to the traditional approaches to solving DOPs where the target is to track the moving global optimum, we aimed at finding robust optimal solutions over time, which opens up a new perspective of solving DOPs referred to as robust optimization over time (ROOT). We proposed a generic framework as well as a robustness measure in the time domain for ROOT. An instantiation of the framework has also been proposed, where a PSO algorithm is employed as optimizer, an RBF model is adopted for estimating fitness in the past, and an AR model for predicting future fitness. The instantiation is empirically evaluated on three benchmarks for ROOT in comparison to their TMO counterparts.

From the simulation results, the following conclusions can be drawn. First, re-using historical information and maintaining the diversity proposed for the TMO approaches are also beneficial for ROOT approaches. However, without explicitly considering the robustness in the time domain, the TMO approaches are not able to find robust solutions over time in general. Second, the proposed framework can reliably find robust solutions over time. For the cases where the TMO approaches can also find robust solutions over time, the proposed framework has significantly better performance in terms of the robustness rate. Third, our proposed framework can also track moving optima, although the moving optima in this case will be robust solutions over time. In fact, ROOT provides an effective way of addressing DOPs having different changing behaviors, e.g., shifting and rotation. We also found that there are conflicting objectives in solving

ROOT problems. More specifically, the robustness rate, the distance to the true optima and the sensitivity of solutions to the changes cannot be optimized simultaneously. Finally, for a better performance of the framework, a lower standard deviation of estimation error of the estimator is essential. For the estimators used in the instantiation studied in this paper, the RBF approximators failed to capture the characteristics of the environments in the past based on the historical data collected online only. However, the simple AR and GRBF predictors performed the prediction task quite well on some problem instances.

As discussed in Sect. 3, the ROOT to DOPs has typically two objectives to take into account, namely, maximizing the average performance over time and minimizing the performance variation. These two objectives can be consistent or conflicting depending on the nature of the problem. In the former case, ROOT can be analyzed and addressed using a single-objective optimization approach [66]. Our future work is to address the conflicting objectives in ROOT by employing a multi-objective approach [67,68] to ROOT for solving DOPs In addition, there is much room for improvement in constructing approximation models for estimating past fitness values. Furthermore, in-depth analysis of the proposed framework is necessary, including more intensive experimental evaluations of the framework on a larger number of benchmark functions.

Acknowledgments This work is partly supported by an EPSRC Grant (no.EP/E058884/1) on "Evolutionary Algorithms for Dynamic Optimisation Problems: Design, Analysis and Applications", the European Union 7th Framework Program under Grant No 247619, a Grant from Honda Research Institute Europe, two National Natural Science Foundation Grants (No. 61028009 and No. 61175065), and the National Natural Science Foundation of Anhui Province (No. 1108085J16).

References

1. Branke J (2002) Evolutionary optimization in dynamic environments. Kluwer, Norwell
2. Jin Y, Branke J (2005) Evolutionary optimization in uncertain environments—a survey. IEEE Trans Evol Comput 9(3):303–317
3. Weise T (2009) Global optimization algorithms—theory and application, 2nd edn. http://www.itweise.de/projects/book.pdf
4. Yu X, Tang K, Chen T, Yao X (2009) Empirical analysis of evolutionary algorithms with immigrants schemes for dynamic optimization. Memet Comput 1(1):3–24
5. Brest J, Zamuda A, Bošković B, Maučec MS, Žumer V (2009) Dynamic optimization using self-adaptive differential evolution. In: Proceedings of the 2009 IEEE congress on evolutionary computation, pp 415–422
6. Yu EL, Suganthan PN (2009) Evolutionary programming with ensemble of explicit memories for dynamic optimization. In: Proceedings of the 2009 IEEE congress on evolutionary computation, pp 431–438
7. Singh HK, Isaacs A, Nguyen TT, Ray T, Yao X (2009) Performance of infeasibility driven evolutionary algorithm (IDEA) on constrained dynamic single objective optimization problems. In: Proceedings of the 2009 IEEE congress on evolutionary computation, pp 3127–3134
8. Yang S, Li C (2010) A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. IEEE Trans Evol Comput 14(6):959–974
9. Yu X, Jin Y, Tang K, Yao X (2010) Robust optimization over time—a new perspective on dynamic optimization problems. In: Proceedings of the 2010 IEEE congress on evolutionary computation, pp 3998–4003
10. Handa H (2006) Fitness function for finding out robust solutions on time-varying functions. In: Proceedings of the 2006 genetic and evolutionary computation conference, pp 1195–1200
11. Bosman PAN (2005) Learning, anticipation and time-deception in evolutionary online dynamic optimization. In: Proceedings of the 2005 genetic and evolutionary computation conference, pp 39–47
12. Powell MJD (1992) The theory of radial basis function approximation in 1990. In: Light W (ed) Advances in numerical analysis, vol 2: wavelets, subdivision algorithms and radial basis functions. Oxfod Univ. Press, London, pp 105–210
13. Liang KH, Yao X, Newton C (2000) Evolutionary search of approximated n-dimensional landscapes. Int J Knowl Based Intell Eng Syst 4(3):172–183
14. Regis RG, Shoemaker CA (2004) Local function approximation in evolutionary algorithms for the optimization of costly functions. IEEE Trans Evol Comput 8(5):490–505
15. Box GEP, Jenkins GM, Reinsel G (1994) Time series analysis: forecasting and control, 3rd edn. Prentice Hall PTR, Upper Saddle River
16. Deb K, Gupta S, Daum D, Branke J, Mall AK, Padmanabhan D (2009) Reliability-based optimization using evolutionary algorithms. IEEE Trans Evol Comput 13(5):1054–1074
17. Beyer HG, Sendhoff B (2007) Robust optimization—a comprehensive survey. Comput Methods Appl Mech Eng 196:3190–3218
18. Branke J (1998) Creating robust solutions by means of an evolutionary algorithm. Parallel Problem Solving from Nature–PPSN V, pp 119–128
19. Greiner H (1996) Robust optical coating design with evolution strategies. Appl Opt 35(28):5477–5483
20. Wiesmann D, Hammel U, Back T (1998) Robust design of multilayer optical coatings by means of evolutionary algorithms. IEEE Trans Evol Comput 2(4):162–167
21. Branke J (2001) Reducing the sampling variance when searching for robust solutions. In: Proceedings of genetic and evolutionary computation conference, pp 235–242
22. Loughlin DH, Ranjithan SR (1999) Chance-constrained genetic algorithms. In: Proceedings of genetic and evolutionary computation conference, pp 369–376
23. Jin Y (2002) Fitness approximation in evolutionary computation—a survey. In: Proceedings of genetic and evolutionary computation conference, pp 1105–1112, New York
24. Jin Y (2011) Surrogate-assisted evolutionary computation: recent advances and future challenges. Swarm Evol Comput 1(2):61–70
25. Paenke I, Branke J, Jin Y (2006) Efficient search for robust solutions by means of evolutionary algorithms and fitness approximation. IEEE Trans Evol Comput 10(4):405–420
26. Lim D, Ong YS, Jin Y, Sendhoff B, Lee BS (2006) Inverse multi-objective robust evolutionary optimization. Genet Program Evol Mach 7(4):383–404
27. Chen XS, Ong YS, Lim MH, Tan KC (2011) A multi-facet survey on memetic computation. IEEE Trans Evol Comput 15(5):591–607
28. Ong YS, Lim MH, Chen XS (2010) Research frontier: memetic computation—past, present & future. IEEE Comput Intell Mag 5(2):24–36

29. Ong YS, Nair PB, Lum KY (2006) Max-min surrogate-assisted evolutionary algorithm for robust design. IEEE Trans Evol Comput 10(4):392–404
30. Le MN, Ong YS, Menzel S, Jin Y, Sendhoff B (2012) Evolution by adapting surrogates. Evol Comput (accepted)
31. Le MN, Ong YS, Jin Y, Sendhoff B (2012) A unified framework for symbiosis of evolutionary mechanisms with application to water clusters potential model design. IEEE Comput Intell Mag 7(1):20–35
32. Le MN, Ong YS, Jin Y, Sendhoff B (2009) Lamarckian memetic algorithms: local optimum and connectivity structure analysis. Memet Comput J 1(3):175–190
33. Branke J (1999) Memory enhanced evolutionary algorithms for changing optimization problems. In: Proceedings of the 1999 IEEE congress on evolutionary computation, vol 3, pp 1875–1882
34. Yang S (2007) Explicit memory schemes for evolutionary algorithms in dynamic environments. In: Yang S, Ong Y-S, Jin Y (eds) Evolutionary computation in dynamic and uncertain environments, chapter 1. Springer, Berlin, pp 3–28
35. Goldberg DE, Smith RE (1987) Nonstationary function optimization using genetic algorithms with dominance and diploidy. In: Grefenstette JJ (ed) Genetic algorithms. Lawrence Erlbaum, Mahwah, pp 59–68
36. Hadad BS, Eick CF et al (1997) Supporting polyploidy in genetic algorithms using dominance vectors. In: Angeline PJ (ed) Evolutionary programming. LNCS, vol 1213. Springer, Berlin, pp 223–234
37. Lewis J, Hart E, Ritchie G (1998) A comparison of dominance mechanisms and simple mutation on nonstationary problems. In: Eiben AE, Bäck T, Schoenauer M, Schwefel H-P (eds) Parallel problem solving from nature. LNCS, vol 1498. Springer, Berlin, pp 139–148
38. Ryan C (1997) Diploidy without dominance. In: Alander JT (ed.) Proceedings of 3rd Nordic workshop genetic algorithms, pp 63–70
39. Cobb HG (1990) An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Naval Res Lab, Washington, DC, Tech. Rep. AIC-90-001
40. Jin Y, Sendhoff B (2004) Constructing dynamic test problems using the multi-objective optimization concept. In: Applications of Evolutionary Computing. LNCS, vol 3005. Springer, Berlin, pp 525–536
41. Yang S, Tinós R (2007) A hybrid immigrants scheme for genetic algorithms in dynamic environments. Int J Autom Comput 4(3):243–254
42. Yu X, Tang K, Yao X (2008) An immigrants scheme based on environmental information for genetic algorithms in changing environments. In: Proceedings of the 2008 IEEE congress on evolutionary computation, pp 1141–1147
43. Branke J, Kaußler T, Schmidt C, Schmeck H (2000) A multipopulation approach to dynamic optimization problems. In: Adaptive computing in design and manufacturing 2000. LNCS. Springer, Berlin
44. Ursem RK (2000) Multinational GA optimization techniques in dynamic environments. In: Whitley D et al (eds) Proceedings of genetic and evolutionary computation conference, pp 19–26
45. Simpson T, Mauery T, Korte J, Mistree F (1998) Comparison of response surface and Kriging models for multidisciplinary design optimization. Technical Report 98-4755, AIAA
46. Jin Y (2005) A comprehensive survey of fitness approximation in evolutionary computation. Soft Comput 9(1):3–12
47. Sapankevych N Sankar R (2009) Time series prediction using support vector machines: a survey. IEEE Comput Intell Mag 4(2):24–38
48. MacQueen J (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability, University of California Press, Berkeley, pp 281–297
49. Ye KQ, Li W, Sudjianto A (2000) Algorithmic construction of optimal symmetric Latin hypercube designs. J Stat Plan Inference 90:145–159
50. Chen S, Cowan CFN, Grant PM (1991) Orthogonal least squares learning algorithm for radial basis function networks. IEEE Trans Neural Netw 2(2):302–309
51. Chng ES, Chen S, Mulgrew B (1996) Gradient radial basis function networks for nonlinear and nonstationary time series prediction. IEEE Trans Neural Netw 7(1):190–194
52. Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: Proceedings of the IEEE international conference on neural networks, vol 4, pp 1942–1948
53. Kennedy J, Eberhart RC (2001) Swarm intelligence. Morgan Kaufmann, San Fransisco
54. Parrott D, Li X (2006) Locating and tracking multiple dynamic optima by a particle swarm model using speciation. IEEE Trans Evol Comput 10(4):440–458
55. Bird S, Li X (2006) Enhancing the robustness of a species-based PSO. In: Proceedings of the 2006 IEEE congress on evolutionary computation, pp 843–850
56. Petrowski A (1996) A clearing procedure as a niching method for genetic algorithms. In: Proceedings of the IEEE international conference on evolutionary computation, pp 798–803
57. Clerc M, Kenndy J (2002) The particle swarm—explosion, stability, and convergence in a multidimensional complex space. IEEE Trans Evol Comput 6(1):58–73
58. Bird S, Li X (2007) Using regression to improve local convergence. In: Proceedings of the 2007 IEEE congress on evolutionary computation, pp 592–599
59. Hoare CAR (1962) Quicksort. Comput J 5(1):10–15
60. Goloub GH, van Loan CF (1996) Matrix computations, 3rd edn. The John Hopkins Press, Baltimore
61. Press WH, Teukolsky SA, Vetterling WT, Flannery BP (1992) Numerical recipes in C: the art of scientific computing, 2nd edn. Cambridge Univ. Press, Cambridge
62. Li C, Yang S, Nguyen TT, Yu EL, Yao X, Jin Y, Beyer H-G, Suganthan PN (2008) Benchmark generator for CEC'2009 competition on dynamic optimization, Technical Report 2008, Department of Computer Science, University of Leicester, UK
63. Salomon R (1996) Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions: A survey of some theoretical and practical aspects of genetic algorithms. BioSystems 39(3):263–278
64. Hu X, Eberhart RC (2002) Adaptive particle swarm optimization: Detection and response to dynamic systems. In: Proceedings of the 2002 IEEE congress on evolutionary computation, pp 1666–1670
65. Richter H (2009) Detecting change in dynamic fitness landscapes. In: Proceedings of the 2009 IEEE congress on evolutionary computation, pp 1613–1620
66. Fu H, Sendhoff B, Tang K, Yao X (2012) Characterizing environmental changes in robust optimization over time. Congress on Evolutionary Computation
67. Jin Y, Sendhoff B (2003) Trade-off between performance and robustness: an evolutionary multiobjective approach. In: Proceedings of second international conference on evolutionary multi-criteria optimization. LNCS, vol 2632. Springer, Berlin, pp 237–251, Faro
68. Jin Y, Sendhoff B (2009) A systems approach to evolutionary multi-objective structural optimization and beyond. IEEE Comput Intell Mag 4(3):62–76