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

Finding Robust Solutions to Dynamic Optimization Problems

Haobo Fu, Bernhard Sendhoff, Ke Tang, Xin Yao

2013

Preprint:

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

Finding Robust Solutions to Dynamic Optimization Problems

Haobo Fu[1], Bernhard Sendhoff[2], Ke Tang[3], and Xin Yao[1]

[1] CERCIA, School of Computer Science, University of Birmingham, UK
[2] Honda Research Institute Europe, Offenbach, DE
[3] Joint USTC-Birmingham Research Institute in Intelligent Computation and Its Applications, School of Computer Science and Technology, University of Science and Technology of China, CN

Abstract. Most research in evolutionary dynamic optimization is based on the assumption that the primary goal in solving Dynamic Optimization Problems (DOPs) is Tracking Moving Optimum (TMO). Yet, TMO is impractical in cases where keeping changing solutions in use is impossible. To solve DOPs more practically, a new formulation of DOPs was proposed recently, which is referred to as Robust Optimization Over Time (ROOT). In ROOT, the aim is to find solutions whose fitnesses are robust to future environmental changes. In this paper, we point out the inappropriateness of existing robustness definitions used in ROOT, and therefore propose two improved versions, namely *survival time* and *average fitness*. Two corresponding metrics are also developed, based on which *survival time* and *average fitness* are optimized respectively using population-based algorithms. Experimental results on benchmark problems demonstrate the advantages of our metrics over existing ones on robustness definitions *survival time* and *average fitness*.

Keywords: Evolutionary Dynamic Optimization, Robust Optimization Over Time, Population-Based Search Algorithms

1 Introduction

Applying population-based search algorithms to solving Dynamic Optimization Problems (DOPs) has become very active [6, 13] as most real-world optimization problems are subject to environmental changes. DOPs deal with optimization problems whose specifications change over time, and the algorithm for DOPs needs to react to those changes during the optimization process as time goes by [9]. So far, most research on DOPs falls into the category of Tracking Moving Optimum (TMO) [2, 12, 11, 8]. Recently, a more practical way of formulating DOPs, namely Robust Optimization Over Time (ROOT), has been proposed [14, 5, 7].

A DOP is usually represented as a dynamic fitness function $F(\overrightarrow{X}, \alpha(t))$, where $\overrightarrow{X}$ stands for the design variable and $\alpha(t)$ is the time-dependent problem parameters. $\alpha(t)$ can change continuously or discretely, and is often considered to be

deterministic at any time point. In this paper, we investigate the case where $\alpha(t)$ changes discretely. Hereafter, we use $F_t(\overrightarrow{X})$ to represent $F(\overrightarrow{X}, \alpha(t))$ for short. Briefly speaking, the objective in TMO is to optimize the current fitness function, while in ROOT solution's current and future fitnesses are both taken into consideration. To be more specific, if the current fitness function is $F_t(\overrightarrow{X})$, TMO is trying to find a solution maximizing [4] F_t, while ROOT aims at the solution whose fitness is not only good for F_t but also stays robust against future environmental changes.

A set of robustness definitions for solutions (A solution is the setting of design variable $\overrightarrow{X}$.) in ROOT have been proposed in [14] and used in [5, 7]. Basically, those definitions consider solution's fitnesses over a time period, either the average of them or the variance. However, those definitions suffer from the following problems:

- All these robustness definitions are dependent on a fitness threshold parameter v, the setting of which requires the information of optimal solution in terms of current fitness at any time point. This limits the practical use of those robustness definitions, as most often the optimal solution for any time point is not known in real-world DOPs.
- A solution is considered 'robust' only if its fitness stays above the threshold parameter v after an environmental change without any constraint on solution's current fitness. This might be inappropriate as robust solutions can have very bad fitnesses for current fitness function. This inappropriateness is reflected in the poor average fitness of robust solutions in the experimental results in [7].
- Robustness definitions based on the threshold parameter v only measure one aspect of robust solutions for DOPs. For example, solutions which have good average fitness over a certain time window could also be considered robust without any constraint on the fitness at any time point. Besides, it is difficult to incorporate the threshold parameter v into the algorithm mainly because the setting of v requires the information of optimal solution at any time point. Algorithms have to know what kind of robust solutions in ROOT they are searching for, just as the distribution information of disturbances is informed to the algorithm in traditional robust optimization [10, 1].

To the best of our knowledge, the only algorithm available for ROOT in the literature is from [7]. An algorithm framework which contains an optimizer, a database, an approximator and a predictor, was proposed in [7]. The basic idea is to average solution's fitness over the past and the future. To be more specific, the optimizer in the framework searches solutions based on a metric [5] which is the average over solution's previous, current and future fitnesses. Solution's previous fitness is approximated using previously evaluated solutions which are stored in the database, while solution's future fitness is predicted based

[4] Without loss of generality, we consider maximization problems in this paper.

[5] A metric is a function which assigns a scalar to a solution to differentiate good solutions from bad ones.

on its previous and current fitnesses using the predictor. The construction of the framework is intuitively sensible for ROOT. However, the metric suffers from two main problems. Firstly, the metric does not incorporate the information of robustness definitions. Therefore, the optimizer does not really know what kind of robust solutions it is searching for. Secondly, estimated fitnesses (either previous or future fitnesses) are used in the metric without any consideration of the accuracy of the estimator (approximator or predictor). This is inappropriate as reliable estimations should be favoured in the metric. For example, if two solutions have the same metric value, the one with more reliable estimations should be considered better than the other.

This paper thus tries to overcome the shortcomings mentioned above regarding existing work for ROOT by first developing two robustness definitions, namely *survival time* and *average fitness*, and a corresponding performance measurement for ROOT. New metrics, based on which *survival time* and *average fitness* are optimized respectively using population-based algorithms, are also proposed. Specially, our metrics incorporate the information of robustness definitions and take estimator's estimation error into consideration. The remainder of the paper is structured as follows. Section 2 presents the robustness definitions *survival time* and *average fitness* in ROOT. After that, one performance measurement is suggested comparing algorithm's ability in finding robust solutions in ROOT. The new metrics are then described in Section 3. Experimental results are reported in Section 4 with regard to performances of the old metric in [7] and our newly proposed metrics on our performance measurement for ROOT. Finally, conclusions and future work are discussed in Section 5.

2 Robustness Definitions and Performance Measurement in ROOT

A DOP is different from a static optimization problem only if the DOP is solved in an on-line manner [6, 9], i.e., the algorithm for DOPs has to provide solutions repeatedly as time goes by. Suppose at time t, the algorithm comes up with a solution $\overrightarrow{X}_t$. The robustness of solution $\overrightarrow{X}_t$ can be defined as:

- the *survival time* F^s equalling to the maximal time length from time t during which the fitness of solution $\overrightarrow{X}_t$ stays above a pre-defined fitness threshold δ:

$$F^s(\overrightarrow{X}, t, \delta) = max\{0 \cup \{l | F_i(\overrightarrow{X}) \geq \delta,\ \forall i,\ t \leq i \leq t + l\}\}, \tag{1}$$

- or alternatively the *average fitness* F^a over a pre-defined time window T from time t:

$$F^a(\overrightarrow{X}, t, T) = \frac{1}{T} \sum_{i=t}^{t+T-1} F_i(\overrightarrow{X}). \tag{2}$$

Both robustness definitions (*survival time* F^s and *average fitness* F^a) do not require the information of optimal solution at any time point, and thus are not restricted to academic studies. For *survival time* F^s, the fitness threshold δ places a constraint on solution's current fitness, which is not satisfied in robustness definitions used in [7]. More importantly, our robustness definitions have user-defined parameters (fitness threshold δ and time window T), which makes it easy to incorporate them into algorithms.

We would like to make a clear distinction between robustness definitions of solutions in ROOT and performance measurements for ROOT algorithms. As a DOP should be solved in an on-line manner and algorithms have to provide solutions repeatedly, algorithms should not be compared just at one time point but across the whole time period. As we consider discrete-time DOPs in this paper, a DOP can be represented as a sequence of static fitness functions $(F_1, F_2, ..., F_N)$ during a considered time interval $[t_0, t_{end})$. Given the robustness definitions in Equation 1 and 2, we could define ROOT performance measurement for time interval $[t_0, t_{end})$ as follows:

$$Performance^{ROOT} = \frac{1}{N}\sum_{i=1}^{N} E(i), \tag{3}$$

where $E(i)$ is the robustness (either *survival time* F^s or *average fitness* F^a) of the solution determined by the algorithm during the time of F_i.

It should be noted that performance measurement for ROOT proposed here is dependent on parameter settings, being either δ if *survival time* F^s is investigated or T if *average fitness* F^a is employed. Therefore, in order to compare algorithms' ROOT abilities comprehensively, results should be reported under different settings of δ or T.

3 New Metrics for Finding Robust Solutions in ROOT

A metric for finding robust solutions in ROOT was proposed in [7], which takes the form $\sum_{i=t-p}^{t+q} F_i(\overrightarrow{X})$ when the current time is t, where p and q are two parameters to control how many time steps looking backward and forward respectively. As discussed in Section 1, the metric does not incorporate the information of robustness definition, and the estimation error is not taken into consideration. To address the two problems, we propose new metrics in the following. As our new metrics take robustness definitions into consideration, we describe the new metrics in the context of *survival time* F^s and *average fitness* F^a respectively.

3.1 Metric for Robustness Definition: *Survival Time*

If we restrict that the metric to optimize *survival time* F^s is a function of solution's current and future fitnesses and user-defined fitness threshold δ, we can

define the metric $\hat{F}^s$ as follows:

$$\hat{F}^s(\overrightarrow{X}, t) = \begin{cases} F_t(\overrightarrow{X}) & \text{if } F_t(\overrightarrow{X}) < \delta, \\ \delta + w * \hat{l} & \text{otherwise,} \end{cases} \tag{4}$$

where $F_t(\overrightarrow{X})$ is the current fitness of solution $\overrightarrow{X}$, and $\hat{l}$ is used to represent the number of consecutive fitnesses which are no smaller than δ starting from the beginning of the fitness sequence $(\hat{F}_{t+1}(\overrightarrow{X}), ..., \hat{F}_{t+L}(\overrightarrow{X}))$. $\hat{F}_{t+i}(\overrightarrow{X})$ is the predicted fitness of solution $\overrightarrow{X}$ at time $t+i$, $1 \leq i \leq L$. $\hat{l}$ can be seen as an explicit estimation of solution's *survival time* robustness. As a result, every time the metric $\hat{F}^s$ is calculated, L number of solution's future fitnesses are predicted if $F_t(\overrightarrow{X}) \geq \delta$. w is the weight coefficient associated with the accuracy of the estimator which is used to calculate $\hat{F}_{t+i}(\overrightarrow{X})$, $1 \leq i \leq L$. In this paper, the root mean square error R^{err} is employed as the accuracy measurement, which takes the form:

$$R^{err} = \sqrt{\frac{\sum_{i=1}^{n_t} e_i^2}{n_t}}, \tag{5}$$

where n_t is the number of sample data, and e_i is the absolute difference between the estimated value produced by the estimator and the true value for the ith sample data. In order to make sure that a larger weight is assigned when the corresponding estimator is considered more accurate, w takes an exponential function of R^{err}:

$$w = exp(-\theta * R^{err}), \tag{6}$$

where θ is a control parameter, $\theta \in [0, +\infty)$. The design of metric $\hat{F}^s$ is reasonable in the sense that it takes the form of current fitness if the current fitness is below the fitness threshold δ. On the other hand, if the current fitness is no smaller than δ, $\hat{F}^s$ only depends on $w * \hat{l}$ which is the product of the weight coefficient w and solution's *survival time* robustness estimation $\hat{l}$.

3.2 Metric for Robustness Definition: *Average Fitness*

The design of a metric for optimizing *average fitness* F^a is more straightforward than that for *survival time* F^s. Basically, in order to estimate *average fitness* F^a, solution's future fitnesses are predicted first and then summed together with solution's current fitness. Therefore, if the user-defined time window is T and the current time is t, we have the following metric:

$$\hat{F}^a(\overrightarrow{X}, t) = F_t(\overrightarrow{X}) + \sum_{i=1}^{T-1} (\hat{F}_{t+i}(\overrightarrow{X}) - \theta * R^{err}), \tag{7}$$

where $\hat{F}_{t+i}(\overrightarrow{X})$, θ and R^{err} take the same meaning as those used for the metric $\hat{F}^s$.

With the new metrics developed in Equation 4 and 7, we can have our new algorithms for ROOT by incorporating them into the generic population-based algorithm framework developed in [7]. For more details of the framework, readers can refer to [7].

4 Experimental Study

We conduct two groups of experiments in this section. The objective of the first group is to demonstrate that it is necessary to incorporate the robustness definitions into the algorithm for ROOT. The metric in [7] (denoted as Jin's) is compared with our metrics, i.e., *survival time* and *average fitness*. One true previous fitness and four future predicted fitnesses are used for Jin's metric, the setting of which is reported to have the best performance in [7]. Five future fitnesses are predicted ($L = 5$) for the metric *survival time* $\hat{F}^s$ when the robustness definition is *survival time*. The control parameter θ is set to be 0 in the first group, which means the accuracy of the estimator is not considered temporarily. In the second group, the metrics *survival time* and *average fitness* are investigated with the control parameter θ set to be 0 and 1. The aim is to demonstrate the advantage of making use of estimator's accuracy when calculating the metrics.

4.1 Experimental Setup

Test Problem: All experiments in this paper are conducted on the modified Moving Peaks Benchmark (mMPB). mMPB is derived from Branke's Moving Peaks Benchmark (MPB) [3] by allowing each peak having its own change severities. The reason to modify MPB that way is to make some parts of the landscape change more severely than other parts. Basically, mMPB consists of several peak functions whose height, width and center position change over time. The mMPB can be described as:

$$F_t(\overrightarrow{X}) = \max_{i=1}^{i=m}\{H_t^i - W_t^i * ||\overrightarrow{X} - \overrightarrow{C}_t^i||_2\}, \tag{8}$$

where H_t^i, W_t^i and C_t^i denote the height, width and center of the ith peak function at time t, $\overrightarrow{X}$ is the design variable, and m is the total number of peaks. Besides, the timer t adds 1 after a certain period of time Δe which is measured by the number of fitness evaluations. H_t^i, W_t^i and $\overrightarrow{C}_t^i$ change as follows:

$$\begin{aligned}
H_{t+1}^i &= H_t^i + height_severity^i * N(0,1),\\
W_{t+1}^i &= W_t^i + width_severity^i * N(0,1),\\
\overrightarrow{C}_{t+1}^i &= \overrightarrow{C}_t^i + \overrightarrow{v}_{t+1}^i,\\
\overrightarrow{v}_{t+1}^i &= \frac{s * ((1-\lambda) * \overrightarrow{r} + \lambda * \overrightarrow{v}_t^i)}{\parallel (1-\lambda) * \overrightarrow{r} + \lambda * \overrightarrow{v}_t^i \parallel},
\end{aligned} \tag{9}$$

where N(0,1) denotes a random number drawn from Gaussian distribution with zero mean and variance one. Each peak's height H_t^i and width W_t^i vary according

to its own $height_severity^i$ and $width_severity^i$, which are randomly initialized within $height_severity_range$ and $width_severity_range$ respectively. H_t^i and W_t^i are constrained in the range $[30, 70]$ and $[1, 12]$ respectively. The center $\overrightarrow{C}_t^i$ is moved by a vector $\overrightarrow{v}^i$ of length s in a random direction ($\lambda = 0$) or a direction exhibiting a trend ($\lambda > 0$). The random vector $\overrightarrow{r}$ is created by drawing random numbers in $[-0.5, 0.5]$ for each dimension and then normalizing its length to s. The settings of mMPB are summarized in Table 1.

Table 1: Parameter settings of the mMPB benchmark

number of peaks, m	5
change frequency, Δe	2500
number of dimensions, D	2
search range	$[0, 50]$
height range	$[30, 70]$
initial height	50
width range	$[1, 12]$
initial width	6
$height_severity_range$	$[1, 10]$
$width_severity_range$	$[0.1, 1]$
trend parameter, λ	1
scale parameter, s	1

In our experiments, we generate 150 consecutive fitness functions with a fixed random number generator. All the results presented are based on 30 independent runs of algorithms with different random seeds.

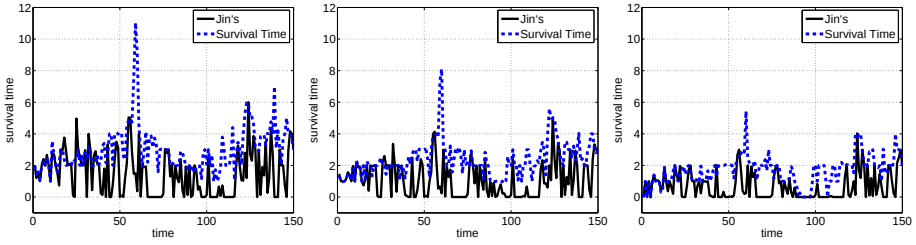
Parameter Settings: We adopt a simple PSO algorithm as the optimizer in this paper. The PSO algorithm used in this paper takes the constriction version. For details of the PSO algorithm, readers are advised to refer to [4]. The swarm population size is 50. The constants c_1 and c_2, which are used to bias particle's attraction to local best and global best, are both set to be 2.05, and therefore the constriction factor χ takes a value 0.729844. The velocity of particles are constricted within the range $[-V_{MAX}, V_{MAX}]$. The value of V_{MAX} is set to be the upper bounds of the search range, which is 50 in our case.

We use the Autoregressive (AR) model for the prediction task. An AR model of order ψ takes the form $Y_t = \epsilon + \sum_{i=1}^{\psi} \eta_i * Y_{t-i}$ where ϵ is the white noise and Y_t is the time series data at time t. We use the least square method to estimate AR model parameters $\overrightarrow{\eta}$ ($\overrightarrow{\eta} = (\eta_1, \eta_2, ...\eta_\psi)$). The parameter ψ is set to be 5 and the latest time series of length 15 are used as the training data. If AR model accuracy is considered, the first 12 time steps are chosen as the training data, and the latest 3 time steps are used to calculate R^{err}. We omit the process of approximating solution's previous fitness but use solution's true previous fitness for both the algorithm in [7] and our algorithms. The reasons are we would like to exclude the effects of approximation error but focus on the effects of prediction error on the metrics, and also it is relatively easy to

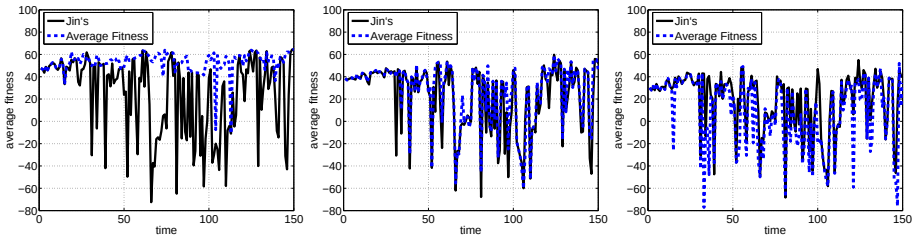
approximate solution's previous fitness given enough historical data, which is usually available in population-based algorithms.

4.2 Simulation Results

The results of the first group experiment are plotted in Fig. 1. In Fig. 1(a), (b), (c) and (d), we can see that the results achieved by our metrics with $\theta = 0$ are generally above those achieved by Jin's metric. This is mainly because our metrics take the corresponding robustness definitions into consideration, and therefore are better at capturing user's preferences of robustness. Our metrics have similar results with Jin's in Fig. 1(e) and (f). This is because by setting T equal to 4 or 6, our metrics happen to have similar forms to Jin's metric. All these results are further summarized in Table 2.



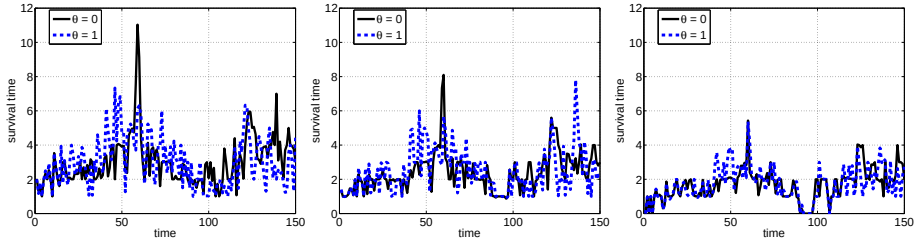
(a) Fitness threshold $\delta = 40$ (b) Fitness threshold $\delta = 45$ (c) Fitness threshold $\delta = 50$



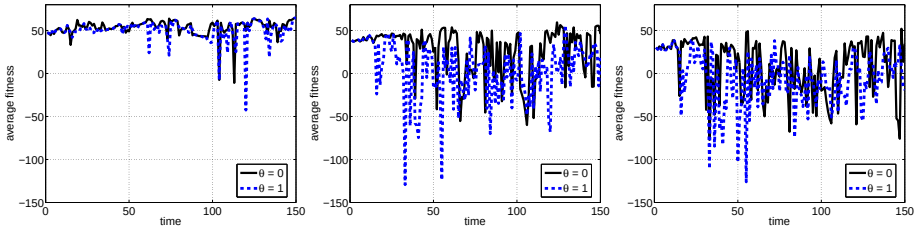
(d) Time window $T = 2$ (e) Time window $T = 4$ (f) Time window $T = 6$

Fig. 1: The averaged robustness over 30 runs for each time step, produced by Jin's metric and our metrics (θ is set to be 0) under robustness definitions of *survival time* F^s and *average fitness* F^a with different settings of δ and T respectively.

The results of the second group experiment are plotted in Fig. 2. The advantage of incorporating estimator's accuracy into metrics has been confirmed in results for *survival time* F^s. This may due to the fact that R^{err} is in accordance with the accuracy in calculating *survival time* estimation $\hat{l}$. However, we can see a performance degrade in making use of estimator's accuracy in the results for *average fitness* F^a. The means R^{err} may not be a good indicator of estimator's accuracy in predicting solution's future fitness. All these results are further summarized in Table 2.



(a) Fitness threshold $\delta = 40$ (b) Fitness threshold $\delta = 45$ (c) Fitness threshold $\delta = 50$



(d) Time window $T = 2$ (e) Time window $T = 4$ (f) Time window $T = 6$

Fig. 2: The averaged robustness over 30 runs for each time step, produced by our metrics when θ is set to be 0 and 1 under robustness definitions of *survival time* F^s and *average fitness* F^a with different settings of δ and T respectively.

5 Conclusions and Future Work

In this paper, we pointed out the inappropriateness of existing robustness definitions in ROOT and developed two new definitions *survival time* F^s and *average fitness* F^a. Moreover, we developed two novel metrics based on which population-based algorithms search for robust solutions in ROOT. In contrast with the metric in [7], our metrics not only take robustness definitions into consideration but also make use of estimator's accuracy.

From the simulation results, we can arrive at that it is necessary to incorporate the information of robustness definitions into the algorithm for ROOT. In other words, the algorithm has to know what kind of robust solutions it is searching for. Secondly, estimator's accuracy can have a large influence on algorithm's performance, and it is important to develop appropriate accuracy measure considering the robustness to be maximized in ROOT.

For the future work, the variance of solution's future fitnesses can be considered as a second objective, and existing multi-objective algorithms can be adapted for it. Also, in what way estimation models should interact with search algorithms is still an open question in ROOT, as solution's future fitnesses are considered in ROOT and prediction task is inevitable.

Table 2: Performance measurement in Equation 3 of investigated algorithms (standard deviation in bracket). Wilcoxon rank sum tests at a 0.05 significance level are conducted between every two of the three algorithms. Significance is indicated in **boldness** for the first and the second, star $*$ for the second and the third and underline for the first and the third.

Algorithms	$\delta = 40$	$\delta = 45$	$\delta = 50$	$T = 2$	$T = 4$	$T = 6$
Jin's	1.53(0.08)	1.11(0.06)	0.69(0.05)	25.32(1.20)	22.20(1.08)	**18.46**(1.06)
Ours ($\theta = 0$)	**3.02**(0.05)	**2.39**(0.05)	**1.69**(0.03)	**53.48***(0.38)	**26.99***(1.12)	8.82*(1.11)
Ours ($\theta = 1$)	3.01(0.08)	2.49*(0.05)	1.72*(0.04)	50.15(0.64)	4.91(1.81)	-5.26(1.98)

References

1. H.G. Beyer and B. Sendhoff. Robust optimization-a comprehensive survey. *Computer methods in applied mechanics and engineering*, 196(33-34):3190–3218, 2007.
2. T. Blackwell, J. Branke, and X. Li. Particle swarms for dynamic optimization problems. *Swarm Intelligence*, pages 193–217, 2008.
3. J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 1999.
4. M. Clerc and J. Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, 2002.
5. H. Fu, B. Sendhoff, K. Tang, and X. Yao. Characterizing environmental changes in robust optimization over time. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012.
6. Y. Jin and J. Branke. Evolutionary optimization in uncertain environments-a survey. *Evolutionary Computation, IEEE Transactions on*, 9(3):303–317, 2005.
7. Y. Jin, K. Tang, X. Yu, B. Sendhoff, and X. Yao. A framework for finding robust optimal solutions over time. *Memetic Computing*, pages 1–16, 2012.
8. C. Li and S. Yang. A general framework of multipopulation methods with clustering in undetectable dynamic environments. *Evolutionary Computation, IEEE Transactions on*, 16(4):556–577, 2012.
9. T.T. Nguyen, S. Yang, and J. Branke. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6:1–24, 2012.
10. I. Paenke, J. Branke, and Y. Jin. Efficient search for robust solutions by means of evolutionary algorithms and fitness approximation. *Evolutionary Computation, IEEE Transactions on*, 10(4):405–420, 2006.
11. P. Rohlfshagen and X. Yao. Dynamic combinatorial optimisation problems: an analysis of the subset sum problem. *Soft Computing*, 15(9):1723–1734, 2011.
12. A. Simões and E. Costa. Prediction in evolutionary algorithms for dynamic environments using markov chains and nonlinear regression. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 883–890. ACM, 2009.
13. S. Yang, Y. Jin, and Y.S. Ong. *Evolutionary Computation in Dynamic and Uncertain Environments*. Springer-Verlag, Berlin, Heidelberg, 2007.
14. X. Yu, Y. Jin, K. Tang, and X. Yao. Robust optimization over time–A new perspective on dynamic optimization problems. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–6. IEEE, 2010.