FINDING ROBUST SOLUTIONS AGAINST ENVIRONMENTAL CHANGES

by

HAOBO FU

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
May 2014

University of Birmingham Research Archive

e-theses repository

This unpublished thesis/dissertation is copyright of the author and/or third parties. The intellectual property rights of the author or third parties in respect of this work are as defined by The Copyright Designs and Patents Act 1988 or as modified by any successor legislation.

Any use made of information contained in this thesis/dissertation must be in accordance with that legislation and must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the permission of the copyright holder.

ABSTRACT

Many real world problems can be formalized as optimisation problems. Yet, the environment in the real world is mostly changing over time, which makes the optimisation problems change over time.

This thesis is dedicated to the problem of finding robust solutions in changing environments. By robustness, we mean that a solution not only is good for the current environment but also maintains its performance after future environmental changes. The problem of finding such robust solutions repeatedly over time is termed as Robust Optimisation Over Time (ROOT).

The first contribution of this thesis is a formal definition of ROOT problems and two robustness definitions of solutions to ROOT. The second contribution of this thesis is two benchmarks developed for ROOT. The third contribution of this thesis is an algorithm framework for finding solutions repeatedly over time to ROOT problems. Finally, the fourth contribution of this thesis is a dynamic handling strategy, which is used to initialize a population of solutions right after an environmental change when evolutionary algorithms are used for ROOT problems.

I would like to dedicate this thesis to my loving parents Weiguo Fu and Hongxing Huang.

Acknowledgements

I would like to express my deep gratitude to my supervisor Prof. Xin Yao for his incredible patience, encouragement, supervision, and dedication. I learned a lot from every conversation, discussion, email, call, and meeting with Xin. During my PhD study, Xin always pushed me to think deeper, which at the beginning I was somewhat reluctant to but later on I found profoundly beneficial. The experience of working with Xin is invaluable. Also, I am indebted to my co-supervisor Prof. Bernhard Sendhoff from the Honda Research Institute Europe for his insightful advice and discussion along the way, and to the Honda Research Institute Europe itself for funding my PhD study.

I am further grateful to my thesis group members Prof. Jon Rowe and Dr. Shan He for their criticisms and suggestions on my research during every thesis group meeting, all of which have greatly improved the thesis and my way of doing research.

Special thanks are given to Prof. Yaochu Jin and Dr. Shan He for agreeing to read my thesis and be my viva examiners.

I would also like to thank some of my excellent teachers, colleagues, and friends for their remarkable collaboration, discussion, and suggestion on the research: Prof. Ke Tang, Prof. Huanhuan Chen, Prof. Yinan Guo, Dr. Leandro Minku, Dr. Peter Lewis, Dr. Ben Jones, Dr. Lisa Schramm, Dr. James Whitacre, Dr. Trung Thanh Nguyen, Dr. Yuan Shen, Dr. Yi Mei, Dr. Xin Yu, Dr. Fei Peng, Dr. Rodrigo Soares, and Dr. Jakramate Bootkrajang.

I am also happy to have shared my PhD life with some of my awesome friends: Dr. Fubiao Xia, Dr. Dongsheng He, Matthew Kent, Wei Liu, Dr. Jinlong Li, Dr. Thomas Weise, Dr. Pu Wang, Dr. Rui

Wang, Dr. Xiaoning Peng, Dr. Shuo Wang, Dr. Guanzhou Lu, Dr. Vero Arriola, Esra Zaghoul, Gurchetan Grewal, Dr. Mengyao Wang, Fengzhen Tang, Jingyi Tan, Xiaofen Lu, and Ruiwen Zhang.

Finally, I wish to thank my great parents Weiguo Fu and Hongxing Huang for their consistent support, kindness, encouragement, and love. This thesis is dedicated to them.

Contents

1 Introduction **1**

1.1 Background 1

1.2 Research Questions 3

1.2.1 What is ROOT? 3

1.2.2 How to Benchmark ROOT? 4

1.2.3 How to Find Solutions to ROOT Problems? 4

1.2.4 How to, from the Perspective of an EA, Handle Environmental Changes in ROOT? 5

1.3 Outline of the Thesis 6

1.4 Contributions of the Thesis 8

1.5 Publications Resulting from the Thesis 9

2 Literature Review on Evolutionary Dynamic Optimisation **11**

2.1 Tracking Moving Optimum 13

2.1.1 Problem Definition of Tracking Moving Optimum 13

2.1.2 EAs for Tracking Moving Optimum Problems 14

2.2 Adaptation and Optimisation 21

2.2.1 Problem Definition of Adaptation and Optimisation 21

2.2.2 EAs for Adaptation and Optimisation Problems 22

2.3 DOPs with Time-linkage 24

2.3.1 Problem Definition of DOPs with Time-linkage 24

2.3.2 EAs for DOPs with Time-linkage 25

2.4 ROOT and Similar Motivations 26

2.5 Differences Between ROOT and Traditional Evolutionary Robust Optimisation 29

2.6 Summary 30

3 Motivation, Problem Definition, and Problem Difficulties of Robust Optimisation Over Time 31

3.1 Motivation of ROOT 31
3.2 A Formal Problem Definition of ROOT 33
3.3 Two Robustness Definitions: ‘Average Fitness’ and ‘Survival Time’ 34
3.4 Assumptions for ROOT Problems Investigated in This Thesis . . 37
3.5 Problem Difficulties of ROOT 39
3.5.1 Difficulties in Predicting a Solution’s Future Fitness in ROOT 39
3.5.2 Difficulties in Evolving a Population Based on Inaccurate Information in ROOT 40
3.6 Summary 40

4 Benchmarking Robust Optimisation Over Time 42

4.1 Existing DOP Benchmarks and Their Problems for ROOT 43
4.1.1 Difficulties in Calculating the Absolute Best ‘Average Fitness’ 44
4.1.2 Difficulties in Calculating the Absolute Best ‘Survival Time’ 46
4.2 Tunable Benchmark Problems for ROOT 47
4.2.1 Baseline Fitness Functions 48
4.2.2 Dynamics for the Baseline Fitness Functions 50
4.2.3 Relationship Between the Benchmark Parameters and ROOT Problem Difficulties 53
4.2.4 Calculating the Absolute Best ‘Average Fitness’ in RMPB-I 54
4.2.5 Calculating the Absolute Best ‘Survival Time’ in RMPB-II 55
4.3 Behaviour of Existing Methods for ROOT 57
4.3.1 Experimental Settings 57
4.3.1.1 Test Problems 57
4.3.1.2 Methods Investigated for ROOT Problems 58
4.3.1.3 Performance Measurement 60
4.3.2 Experimental Results 61
4.3.2.1 Results for Maximizing ‘Average Fitness’ 61
4.3.2.2 Results for Maximizing ‘Survival Time’ 62

4.3.2.3 Gaps Between the Absolute Best and the Results obtained by Investigated Methods 63
4.4 Summary 65

5 Learning and Optimizing in Robust Optimisation Over Time 67
5.1 Existing Methods for ROOT 67
5.2 A Novel Algorithm Framework for ROOT 69
5.2.1 Ensemble Learning for ROOT 69
5.2.2 Optimizing for ROOT 71
5.2.2.1 Maximizing ‘Average Fitness’ in ROOT 72
5.2.2.2 Maximizing ‘Survival Time’ in ROOT 72
5.2.3 An Optimizing and Ensemble Learning Framework for ROOT 74
5.2.4 Computational Complexity 76
5.3 Experimental Studies 79
5.3.1 Experimental Settings 79
5.3.1.1 Test Problems 79
5.3.1.2 Performance Measurement 82
5.3.1.3 The Setting of PSO 82
5.3.1.4 Learning Models for Approximation and Prediction 82
5.3.1.5 Methods for Comparison 83
5.3.2 Simulation Results 84
5.3.2.1 Peer Methods Versus OEL 85
5.3.2.2 Single Model Versus OEL 87
5.3.2.3 The Absolute Best Performance Versus OEL . . . 89
5.3.3 More Discussions on OEL 91
5.3.3.1 Varying the Number of Clusters in the K-means in OEL 91
5.3.3.2 Varying the Length of Fitness Time Series in OEL 91
5.3.3.3 Varying the Problem Parameter T in ROOT . . . 93
5.3.3.4 Varying the Problem Parameter V in ROOT . . 95
5.4 Summary 96

6 Distribution Prediction for Handling Environmental Changes in Robust Optimisation Over Time **98**

6.1 Existing Strategies in EDO for Handling Environmental Changes in DOPs . . . 99

6.2 Distribution Prediction in Robust Optimisation Over Time 101

6.2.1 Estimating Distributions of Good Solutions 102

6.2.2 Prediction of Distributions of Good Solutions 103

6.2.3 Population Initialization 104

6.2.3.1 Initialization for Maximizing 'Average Fitness' . . 105

6.2.3.2 Initialization for Maximizing 'Survival Time' . . . 107

6.2.4 DPS for ROOT 107

6.3 Experimental Studies 108

6.3.1 Experimental Settings 108

6.3.1.1 Test Problems 108

6.3.1.2 Performance Measurement 110

6.3.1.3 ROOT Solver 111

6.3.1.4 Strategies in Comparison 112

6.3.2 Simulation Results 114

6.3.2.1 Results for Maximizing 'Average Fitness' 114

6.3.2.2 Results for Maximizing 'Survival Time' 116

6.3.2.3 Balance of Diversity and Convergence 116

6.3.3 More Discussions on DPS 120

6.3.3.1 Influence of the Time Series Prediction Model in DPS 120

6.3.3.2 Influence of the Parameter ρ_s 121

6.3.3.3 Influence of the Parameter ρ_{init} 122

6.4 Summary 123

7 Conclusions and Future Work **125**

7.1 Conclusions 125

7.2 Future Work 127

Appendices **130**
.1 Proof of Lemma 4.2.1 131
.2 Proof of Lemma 4.2.2 131
.3 Proof of Theorem 4.2.1 132
.4 Proof of Lemma 4.2.3 132
.5 Proof of Lemma 4.2.4 133
.6 Proof of Theorem 4.2.2 134
.7 Surrogate Model for Fitness Approximation 135
.8 Learning Models for Time Series Prediction 135

List of References **138**

List of Figures

3.1 Illustration of a solution's 'average fitness'. 35

3.2 Illustration of a solution's 'survival time'. 37

4.1 One fitness landscape example with 5 peaks along each dimension generated by the baseline fitness function in RMPB-I. 49

4.2 One fitness landscape example with 20 peaks along each dimension generated by the baseline fitness function in RMPB-I. 50

4.3 One fitness landscape example with 5 peaks along each dimension generated by the baseline fitness function in RMPB-II. 50

4.4 One fitness landscape example with 20 peaks along each dimension generated by the baseline fitness function in RMPB-II. 51

4.5 The average performance in Equation 4.18 with one standard deviation errorbar over 30 runs of investigated methods under the robustness definition of 'average fitness' with different settings of time window T, in comparison with the corresponding absolute best performance. 64

4.6 The average performance in Equation 4.18 with one standard deviation errorbar over 30 runs of investigated methods under the robustness definition of 'survival time' with different settings of fitness threshold V, in comparison with the corresponding absolute best performance. 65

5.1 The averaged RMSE over 30 runs at each time step obtained by methods 'AR', 'NN', 'MA', and OEL on TP_{14} with T being 2. . . 88

5.2 The absolute best performance of the robustness 'average fitness' in benchmark TP_{24} and the robustness 'survival time' in benchmark TP_{34} with different settings of T and V respectively, compared to results obtained by OEL. 90

5.3 The averaged performance measure in Equation 5.9 over 30 runs with one standard deviation errorbar under different settings of number of clusters in the K-means when maximizing 'average fitness' with $T = 3$ on TP_{24}. 92

5.4 The averaged performance measure in Equation 5.9 over 30 runs with one standard deviation errorbar under different settings of number of clusters in the K-means when maximizing 'survival time' with $V = 45$ on TP_{34}. 92

5.5 The averaged performance measure in Equation 5.9 over 30 runs with one standard deviation errorbar under different settings of length of the fitness time series when maximizing 'average fitness' with $T = 3$ on TP_{24}. 93

5.6 The averaged performance measure in Equation 5.9 over 30 runs with one standard deviation errorbar under different settings of length of the fitness time series when maximizing 'survival time' with $V = 45$ on TP_{34}. 94

5.7 The averaged performance measure in Equation 5.9 over 30 runs with one standard deviation errorbar under different settings of T when maximizing 'average fitness' on TP_{24}. 94

5.8 The averaged performance measure in Equation 5.9 over 30 runs with one standard deviation errorbar under different settings of V when maximizing 'survival time' on TP_{34}. 95

6.1 The averaged diversity and convergence of the initial population produced by each dynamic handling strategy at the beginning of a time step on TP_{14}. Each data point in the figure is the average over 30 runs and over time step 21 to 100. For each dynamic handling strategy, we plot a data point for every setting of Δe with bigger size for larger Δe. 119

6.2 The averaged performance measure using Equation 6.9 obtained by DPS with different settings of ρ_s and Δe over 30 runs when maximizing 'average fitness' with the time window $T = 1$ on TP_{14}. 122

6.3 The averaged performance measure using Equation 6.9 obtained by DPS with different settings of ρ_{init} and Δe over 30 runs when maximizing 'average fitness' with the time window $T = 1$ on TP_{14}. 123

List of Tables

4.1 Parameter settings of the test problems. 58
4.2 The average performance in Equation 4.18 over 30 runs under the robustness definition of 'average fitness' with different settings of time window T. 62
4.3 The average performance in Equation 4.18 over 30 runs under the robustness definition of 'survival time' with different settings of fitness threshold V. 63

5.1 Parameter settings of the test problems 81
5.2 The averaged performance measure in Equation 5.9 over 30 runs under the robustness definition of 'average fitness' with different settings of time window T (peer methods versus OEL). 86
5.3 The averaged performance measure in Equation 5.9 over 30 runs under the robustness definition of 'survival time' with different settings of fitness threshold V (peer methods versus OEL). 87
5.4 The averaged performance measure in Equation 5.9 over 30 runs under robustness definition of 'average fitness' with different settings of time window T (single model versus OEL). 89
5.5 The averaged performance measure in Equation 5.9 over 30 runs under robustness definition of 'survival time' with different settings of fitness threshold V (single model versus OEL). 89

6.1 Parameter settings of the test problems 111

6.2 The averaged performance measure using Equation 6.9 of different strategies over 30 runs when maximizing ‘average fitness’ with different settings of the time window T and the computational budget Δe. 117

6.3 The averaged performance measure using Equation 6.9 of different dynamic handling strategies over 30 runs when maximizing ‘survival time’ with different settings of the fitness threshold V and the computational budget Δe. 118

6.4 The averaged performance measure using Equation 6.9 obtained by DPS with different time series prediction models over 30 runs when maximizing ‘average fitness’ with different settings of the time window T and the computational budget Δe. 121

6.5 The averaged performance measure using Equation 6.9 obtained by DPS with different time series prediction models over 30 runs when maximizing ‘survival time’ with different settings of the fitness threshold V and the computational budget Δe. 121

List of Abbreviations

AO	Adaptation and Optimisation
AR	Auto-regression
DFF	Dynamic Fitness Function
DMOP	Dynamic Multi-objective Optimisation Problems
DOP	Dynamic Optimisation Problem
DPS	Distribution Prediction Strategy
EA	Evolutionary Algorithm
EDO	Evolutionary Dynamic Optimisation
LPS	Last Population Strategy
MA	Moving Average
MAF	Maximal Average Fitness
MIF	Maximal Intersection Fitness
MS	Memory Strategy
MSE	Mean Square Error
NN	Nearest Neighbourhood
OEL	Optimizing and Ensemble Learning
OPS	Optimum Prediction Strategy

PSO	Particle Swarm Optimizer
RBFN	Radial Basis Function Network
RMPB	Robust Moving Peak Benchmark
RMSE	Root Mean Square Error
ROOT	Robust Optimisation Over Time
RRS	Random Restart Strategy
TMO	Tracking Moving Optimum

CHAPTER 1

INTRODUCTION

This chapter introduces the research topic of this thesis, which is Robust Optimisation Over Time (ROOT), and gives an overview of the whole thesis. The rest of this chapter is organised as follows. Section 1.1 discusses the general background of the research topic ROOT. A number of research questions are then proposed with regard to ROOT in Section 1.2. Section 1.3 outlines the subsequent chapters. Finally, a list of contributions of the thesis on ROOT is presented in Section 1.4.

1.1 Background

Many real world problems can be formalised as optimisation problems. An optimisation problem can be defined as:

Definition 1.1.1. *Given an objective function f that is a mapping from some set $\mathbb{S}$, i.e., a solution space, to the real numbers $\mathbb{R}$: $\mathbb{S} \rightarrow \mathbb{R}$, an optimisation problem is to determine an optimal solution $\boldsymbol{x}^*$ in $\mathbb{S}$ such that, for all $\boldsymbol{x} \in \mathbb{S}$, $f(\boldsymbol{x}^*) \geq f(\boldsymbol{x})$*[1].

In practice, it might be impossible to check whether a determined solution is optimal or not, and an optimisation problem is then about determining a solution with its fitness f as large as possible given limited computational resource.

[1]Maximization problems are considered in this thesis without loss of generality. Minimization problems can be transferred into maximization problems by multiplying the objective fitness function with -1.

A lot of real world optimisation problems are subject to environmental changes over time (Branke [2001], Jin and Branke [2005], Nguyen et al. [2012a]). By environmental changes, we mean changes to the objective fitness function f, the number of decision variables of the solution space $\mathcal{S}$, the constraints that a feasible solution in the solution space $\mathcal{S}$ must satisfy, etc. For instance, in some real world scheduling problems (Barlow and Smith [2008], Yang et al. [2010], Bui et al. [2012]), environmental changes happen over time, such as the arrival of new jobs, the breakdown of communication links, bad weather, etc. In some real world control problems (Ursem et al. [2002], Michalewicz et al. [2007]), the system's state is changing over time, which makes optimal controls vary across time.

If environmental changes to optimisation problems are taken into account explicitly, i.e., new solutions are provided to react to the changes, such optimisation problems are termed as Dynamic Optimisation Problems (DOPs) (Branke [2001], Jin and Branke [2005], Nguyen et al. [2012a]). DOPs consist of different types of problems and can be mainly classified according to the way the decision maker provides solutions to react to environmental changes, the details of which will be discussed in Chapter 2. Assuming that environmental changes occur discretely over time, a DOP can thus be represented using a sequence of fitness functions $(f_1, f_2, ..., f_N)$[1] during a considered time interval $[t_0, t_{end})$. A natural way of providing solutions to such a DOP is determining and implementing[2] a new solution whenever an environmental change happens. Therefore, for a DOP represented as a sequence of fitness functions $(f_1, f_2, ..., f_N)$ during a considered time interval $[t_0, t_{end})$, a sequence of solutions $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N)$, one solution corresponding to one fitness function, has to be determined and implemented.

It is easy to understand that the way of solving a DOP by implementing a new solution every time an environmental change happens may only be justifiable in situations where implementing a solution can be finished instantaneously, compared to the frequency of environmental change, and with little cost (Stroud

[1] In the following of the thesis, we will be discussing only DOPs that can be represented using a sequence of fitness functions over time.

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

[2001], Ursem et al. [2002], Rossi et al. [2008]). In other words, in situations where the implementation of a solution involves human operation (Handa et al. [2005]), resource transportation (Bui et al. [2012]), etc, which all incur a huge cost, it is impossible to implement a new solution every time the environment changes. Based on this consideration, Yu et al. [2010] proposed the idea of finding robust solutions to DOPs. Such a robust solution is aimed to be used for not just one fitness function but a consecutive number of fitness functions over time. In other words, if such a robust solution is implemented, it can be kept in use after environmental changes. The problem of finding such robust solutions repeatedly over time is termed as ROOT (Yu et al. [2010]).

This thesis is dedicated to the research of ROOT. In the next section, a list of research questions with regard to ROOT will be presented, and this thesis tries to answer these questions.

1.2 Research Questions

1.2.1 What is ROOT?

The general idea of ROOT is to repeatedly find solutions whose performance is robust against environmental changes over time (Yu et al. [2010], Jin et al. [2012]). In ROOT, the decision maker is providing new solutions to optimisation problems that are subject to environmental changes. Therefore, ROOT problems can be considered as one type of DOPs, according to the definition of DOPs (Branke [2001], Jin and Branke [2005], Nguyen et al. [2012a]). Yet, the definition of ROOT and the relationship between ROOT problems and other types of DOPs are not quite clear (Yu et al. [2010], Jin et al. [2012]), and one of the reasons is that the robustness of a solution to ROOT problems was not practically motivated.

A deep understanding of the relationship between ROOT problems and other types of DOPs is beneficial. More importantly, a problem definition of ROOT that captures the distinctive feature of ROOT problems, which is finding robust solutions against environmental changes compared to other types of DOPs, is indispensable. The reasons are as follows. Firstly, existing research on other types of DOPs can be transferred to the solving of ROOT problems. Secondly, specific methods can be developed to deal with the distinctive feature of ROOT problems. Finally, people can understand the potential advantages of formulating a DOP as a ROOT problem rather than other types of problems, and hence real world applications of ROOT can be identified easily. Based on the above consideration, we are interested in answering the following research question.

Research Question 1: How to formally define ROOT problems so that the distinctive feature of ROOT problems compared to other types of DOPs can be captured and that real world applications of ROOT can be easily identified?

1.2.2 How to Benchmark ROOT?

Given that the ROOT problem is clearly and formally defined, it is good to have some methods tested and evaluated on ROOT benchmark problems before they are applied to real world ROOT problems. From the test results on benchmark problems, we can understand the strength of different methods on ROOT problems with different characteristics. Also, the difficulties of ROOT problems can be controlled within benchmarks, which can be used to test the scalability of ROOT methods. Since the ROOT problem is one type of DOPs, an immediate question is that whether it is sufficient to test an algorithm's ROOT ability on existing DOP benchmarks. More specially, we are interested in answering the following research question.

Research Question 2: Is it sufficient to test an algorithm's ROOT ability on existing DOP benchmarks? If not, why and how to develop suitable ROOT benchmarks?

1.2.3 How to Find Solutions to ROOT Problems?

The central idea of ROOT is to find solutions whose performance is robust against future environmental changes. By future environmental changes, we mean the changes that the decision maker does not exactly know beforehand until they happen. Therefore, in order to find solutions to ROOT problems, prediction techniques are needed. The prediction techniques are used to predict either what the future (i.e., the future fitness function) will be or what the future fitness of a solution will be. Under mild assumptions, which will be discussed in Section

3.4, the question is how to build such a prediction model for ROOT. Also, once predictions into the future can be made, the question is then how to come up with ROOT solutions based on the predictions. Based on the above discussions, given that in this thesis we are trying to apply Evolutionary Algorithms (EAs) to ROOT problems, we are interested in answering the following research question.

Research Question 3: When trying to find solutions to ROOT problems, where a solution's future fitness matters, under mild assumptions discussed in Section 3.4, how to build a prediction model such that a solution's future fitness can be predicted? Given the prediction of a solution's future fitness, how to design an EA such that it searches for good solutions to ROOT problems?

1.2.4 How to, from the Perspective of an EA, Handle Environmental Changes in ROOT?

When an environmental change happens in DOPs, a simple strategy for an EA to solve DOPs is to restart itself. In this way, DOPs are solved as a sequence of independent optimisation problems (an optimisation problem is defined in Definition 1.1.1). Restarting EAs from scratch whenever an environmental change happens is inefficient for the following reasons. Firstly, for real world DOPs, the fitness function after an environmental change is often related to previous fitness functions (Branke [2001]). Therefore, information gathered in the past is useful for optimizing the present. Secondly, in cases where the environment is changing fast, a restart of EAs won't have enough time to converge to good solutions. Finally, in some DOPs, there exists time-linkage where a solution implemented in the past has an influence on what DOPs look like in the future. Solving such DOPs as a sequence of independent optimisation problems is certainly suboptimal. Therefore, it is important to explicitly handle environmental changes instead of restarting EAs when solving DOPs using EAs.

For the same reasons, it is important to explicitly handle environmental changes in ROOT, from the perspective of EAs developed for ROOT problems, because ROOT problems are one type of DOPs. In other words, we are interested in the following research question:

Research Question 4: When an environmental change happens in a ROOT problem being solved by an EA, instead of restarting the EA, what information can be learned from the past to facilitate the optimizing process for the current? More specifically, instead of randomly initializing a population, how to initialize a promising population in the EA so that useful information can be learned from the past to facilitate the optimizing process for the current?

1.3 Outline of the Thesis

This thesis is trying to answer the research questions listed in Section 1.2, and the rest of the thesis is organised as follows:

Chapter 2 reviews related work on applying EAs to DOPs, which is referred to as Evolutionary Dynamic Optimisation (EDO). The purpose of this chapter is to present a general review of what has been done in terms of solving DOPs using EAs and demonstrate that ROOT presents some new challenges that have not been investigated before in EDO. Three major types of DOPs, in addition to ROOT problems, are identified from EDO. For each of the three major types of DOPs, typical optimizing techniques are discussed, and more importantly the differences between them and ROOT are analysed. Also, previous work on ROOT is briefly reviewed, and similar motivations to ROOT in the literature, which also try to find robust solutions in dynamic environments, are briefly discussed, with an emphasis of their differences to ROOT.

Chapter 3 is dedicated to answering Research Question 1 proposed in Section 1.2.1. Real world DOP applications where it is desirable to have a robust solution that can be used after environmental changes are first identified. Based on the investigation of such real world DOP applications, a practically motivated problem definition of ROOT is given, together with two robustness definitions of solutions to ROOT problems. Assumptions with regard to the ROOT problems investigated in this thesis are made clear. Finally, problem difficulties of ROOT are briefly discussed, especially from the perspective of an EA.

In Chapter 4, we are trying to answer Research Question 2 proposed in Section 1.2.2. Problems of existing DOP benchmarks used for the study of ROOT are first identified, which justify the need to develop specific ROOT benchmark problems. Two new ROOT benchmarks are developed, which aim to test an algorithm's ROOT ability respectively under the two robustness definitions proposed in Section 3.3. Some representative EDO methods are then evaluated on our developed ROOT benchmarks, which demonstrates for the first time that existing EDO methods do not solve ROOT problems well and that there is need to develop more specialized algorithms for ROOT problems.

Chapter 5 is about answering Research Question 3 proposed in Section 1.2.3. Existing methods for ROOT problems are briefly reviewed, with discussions of their potential weaknesses for ROOT problems. A novel algorithm framework is then proposed specially for solving ROOT problems. The algorithm framework consists of an optimizer for searching for ROOT solutions, a component for estimating a solution's previous fitness via building surrogate models, and a component for predicting a solution's future fitness via ensemble learning. Some experiments on benchmark problems, including those developed in Chapter 4, are then conducted to validate the effectiveness of the algorithm framework over existing ROOT methods. Performances with regard to different EDO methods for ROOT problems under different dynamics are discussed.

Chapter 6 addresses Research Question 4 proposed in Section 1.2.4. Existing dynamic handling strategies in EDO, which are used to initialize a new population whenever an environmental change happens, are briefly reviewed, with discussions of their potential weaknesses in different situations. A novel dynamic handling strategy is then developed, which is used to initialize a new population whenever an environmental change happens in ROOT. Within our dynamic handling strategy, distributions of good solutions are predicted by building a learning model on time series of distributions of good solutions over time. The effectiveness of our dynamic handling strategy is evaluated against existing dynamic handling strategies in EDO on ROOT problems with different dynamics and under different budgets of computational resource. Moreover, the diversity and the convergence of the initial population, produced by different dynamic handling strategies right after an environmental change, are analysed, which further explains the success of our dynamic handling strategy over other strategies.

Conclusions of the thesis are given in Chapter 7, with some discussions of the future work on ROOT.

1.4 Contributions of the Thesis

This thesis presents the following list of contributions.

- A practical definition of ROOT problems and two practical definitions of robustness of solutions to ROOT. The unique feature of ROOT, compared to other types of DOPs, is that a ROOT solution is used for a long time period during which environmental changes happen, and this feature is captured in our definition of ROOT. The two robustness definitions have been motivated by investigating real world applications where it is desirable to have a solution in use after environmental changes (Chapter 3).

- Two new types of benchmark problems that are motivated by the inappropriateness of existing DOP benchmarks for the study of ROOT. Representative methods in EDO are evaluated on our proposed ROOT benchmarks, which shows the strengths and weaknesses of different methods in solving ROOT problems with different dynamics. In particular, the real challenges of ROOT problems have been revealed for the first time by the experimental results on our proposed ROOT benchmarks (Chapter 4).

- A novel algorithm framework for finding solutions repeatedly over time to ROOT problems. Within the framework, two new metrics, which are designed respectively for the two robustness definitions of solutions, are used to guide a population-based search algorithm towards optimal solutions. A new mechanism is proposed for building surrogate models in the framework, which are used to estimate a solution's previous fitness. Given a solution's previous and current fitness, we propose to predict a solution's future fitness via an ensemble learning mechanism. The predicted future fitness and the current fitness are used to calculate the corresponding metric of a solution (Chapter 5).

- A novel dynamic handling strategy, in which distributions of good solutions are predicted, for initializing a new population in an EA whenever there is an environmental change in ROOT problems. In our strategy, an explicit learning model is trained on a time series of distributions of good solutions at each previous time step. Distributions of good solutions are then predicted and used to initialize a population whenever there is an environmental change in ROOT problems (Chapter 6).

1.5 Publications Resulting from the Thesis

Refereed or submitted journal papers

- [1] H. Fu, B. Sendhoff, K. Tang, and X. Yao. Robust optimisation over time: Problem difficulties and benchmark problems. *Evolutionary Computation, IEEE Transactions on*, c (accepted)
- [2] H. Fu, L. L. Minku, B. Sendhoff, K. Tang, and X. Yao. Optimizing and learning in robust optimisation over time. *submitted to Cybernetics IEEE Transactions on*, b (submitted)
- [3] H. Fu, H. Chen, B. Sendhoff, K. Tang, and X. Yao. Distribution prediction in robust optimisation over time. *submitted to Evolutionary Computation, IEEE Transactions on*, a (revised and resubmitted)

Published conference papers

- [4] H. Fu, B. Sendhoff, K. Tang, and X. Yao. Characterizing environmental changes in robust optimisation over time. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012
- [5] H. Fu, B. Sendhoff, K. Tang, and X. Yao. Finding robust solutions to dynamic optimisation problems. In *Applications of Evolutionary Computation*, pages 616–625. Springer, 2013
- [6] H. Fu, P. R. Lewis, B. Sendhoff, K. Tang, and X. Yao. What are dynamic optimisation problems? In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC)*. IEEE Press, 2014a. in press
- [7] Y. Guo, M. Chen, H. Fu, and Y. Liu. Find robust solutions over time by two-layer multi-objective optimisation method. In *Proceedings of the 2014*

IEEE Congress on Evolutionary Computation (CEC). IEEE Press, 2014. in press

- [8] H. Fu, P. R. Lewis, and X. Yao. A q-learning based evolutionary algorithm for sequential decision making problems. *In Proceedings of the Workshop "In Search of Synergies between Reinforcement Learning and Evolutionary Computation" at the 13th International Conference on Parallel Problem Solving from Nature (PPSN)*, 2014b. in press

The following is the mapping between the publications and each chapter in the thesis:

- Chapter 2: Publications [4, 5]
- Chapter 3: Publications [1, 5]
- Chapter 4: Publications [1]
- Chapter 5: Publications [2]
- Chapter 6: Publications [3]

CHAPTER 2

LITERATURE REVIEW ON EVOLUTIONARY DYNAMIC OPTIMISATION

DOPs represent a board class of optimisation problems that change over time, and the changes are taken into consideration by the decision maker (Branke [2002], Jin and Branke [2005], Nguyen et al. [2012a]). Usually, a DOP is formalised by a Dynamic Fitness Function (DFF) and how the decision maker provides solutions to the DFF. A DFF is basically a changing fitness function $f(\mathbf{x}, \boldsymbol{\alpha})$, in which $\mathbf{x}$ denotes the decision variables and $\boldsymbol{\alpha}$ represents environmental states that change over time. The terms DFF and environment are used interchangeably in this thesis. ROOT is one type of formalisations of DOPs, and there are other common formalisations of DOPs in the literature. In this chapter, existing work on DOPs using EAs is reviewed, in which we will focus on the methods for different types of DOPs. Particularly, major formalisations of DOPs are discussed respectively, and main features of each major formalisation of DOPs are stressed, which in turn emphasizes the unique feature of ROOT. As it is impossible to cover everything in EDO in one chapter, readers are advised to refer to other resources for other issues in EDO: theoretical work in EDO (Rowe [1999], Rowe [2001], Droste [2002], Droste [2003], Rohlfshagen et al. [2009]), benchmark problems in EDO (Branke [1999], Morrison and De Jong [1999], Jin and Sendhoff [2004], Nguyen and Yao [2009a], Nguyen [2011], Nguyen et al. [2012a]), and performance measures in EDO (Branke [2002], Weicker [2002], Morrison [2003], Nguyen [2011], Nguyen et al. [2012a]).

Assuming that the DFF of a DOP can be represented as a sequence of static fitness functions $(f_1, f_2, ..., f_N)$ during a considered time interval $[t_0, t_{end})$ (one time step corresponds to one static fitness function), i.e., $\boldsymbol{\alpha}$ in a DFF $f(\mathbf{x}, \boldsymbol{\alpha})$ changes discretely over time, we can divide DOPs studied in EDO into the following four categories according to the way the decision maker provides solutions to the DFF. The first type, which is most widely studied so far (Branke [1999], Blackwell et al. [2008], Nguyen and Yao [2009a], Woldesenbet and Yen [2009], Yang and Li [2010], Rohlfshagen and Yao [2011]), is called Tracking Moving Optimum (TMO). Basically, in TMO the decision maker aims to find an optimal solution for the current fitness function and relocate a new optimum once the current fitness function changes into the next one.

The second type of DOPs is called Adaptation and Optimisation (AO) (Bui et al. [2012]). Every time the DFF changes, the decision maker has to provide a new solution. The decision maker has to consider not only the solution's current fitness but also the adaptation cost incurred by adapting the previously implemented solution to the solution for the current fitness function. Therefore, AO problems are intrinsically Dynamic Multiobjective Optimisation Problems (DMOPs)[1] (Bui et al. [2012]), although some work treats AO problems as TMO problems with special constraints (Van de Vonder et al. [2007], Van de Vonder et al. [2008]).

In the third type of DOPs, solutions implemented in the past have an impact on how the DFF changes in the future (Bosman [2005], Nguyen and Yao [2013]), and the decision maker needs to consider a solution's impact on future fitness functions in addition to its fitness for the current fitness function. This kind of problems are called DOPs with time-linkage (Bosman [2005]), and their difficulty lies in how to extract the actual dependence of future fitness functions on solutions implemented previously.

In contrast to the aforementioned three types of DOPs where the decision maker has to implement a new solution every time the DFF changes, the idea of ROOT is to repeatedly find robust solutions whose performance is not only good for the current fitness function but also future fitness functions (Yu et al.

[1]In existing studies for DMOPs using EAs, the main objective is to track the moving Pareto front (Zhou et al. [2011]).

[2010], Fu et al. [2012], Jin et al. [2012], Fu et al. [2013]). The unique feature of ROOT, compared to the aforementioned three types of DOPs, is that once a robust solution is implemented it is used for multiple consecutive fitness functions over time. In other words, it is not necessary to implement a new solution every time the DFF changes in ROOT.

In the following sections, the four main types of DOPs studied in EDO are briefly reviewed, in which we will focus on the problem definition and corresponding methods.

2.1 Tracking Moving Optimum

2.1.1 Problem Definition of Tracking Moving Optimum

Supposing a DFF can be represented as a sequence of static fitness functions $(f_1, f_2, ..., f_N)$ during a considered time interval $[t_0, t_{end})$, the goal of TMO is to find a solution that maximizes the current fitness function at each time step. A TMO problem is defined as:

$$\max \sum_{i=1}^{N} f_i(\mathbf{x}_i), \tag{2.1}$$
$$s.t.\ \mathbf{x}_i\ is\ feasible,\ 1 \leq i \leq N,$$

where $\mathbf{x}_i$ denotes the solution determined at time step i when the DFF takes the form of f_i. $f_i(\mathbf{x}_i)$ is the fitness of solution $\mathbf{x}_i$ at time step i. An implicit assumption of TMO is that maximizing the sum in Equation 2.1 is equivalent to finding a solution that maximizes the current fitness function at every time step. This implicit assumption is not necessarily true in all possible cases since a solution implemented in a previous time step may influence how the DFF changes in later time steps. In other words, maximizing the current fitness function at each time step does not necessarily maximize the sum in Equation 2.1. Another implicit assumption of TMO is that, every time the DFF changes from one fitness function to the next, a new solution has to be implemented. In other words, once solution $\mathbf{x}_i$ is determined at time step i, it has to be implemented into practice at time step i.

It should be noted that any TMO method is supposed to solve a TMO problem in an on-line manner. To be more specific, a TMO method starts when the DFF takes the form of f_1 and determines a solution $\mathbf{x}_1$ within a computational budget Δe. After that, solution $\mathbf{x}_1$ will be implemented into practice. Once the DFF changes from f_1 to f_2, the TMO method will determine a new solution $\mathbf{x}_2$ within the computational budget Δe, and solution $\mathbf{x}_2$ is then implemented into practice. The process repeats till the last fitness function f_N. Also, the computational budget Δe should be generally smaller than the time interval between two successive environmental changes.

2.1.2 EAs for Tracking Moving Optimum Problems

The objective of TMO is to find an optimum for the current fitness function, say f_t, as fast as possible, given historical evaluation information collected while optimizing previous fitness functions f_i, $1 \leq i < t$. Various heuristics have been developed to adapt existing EAs for static optimisation problems for the purpose of TMO:

- *Producing diversity when a change occurs*: Population diversity is generated as soon as the DFF changes. An early example is the hyper-mutation approach (Cobb [1990]), in which the mutation rate is increased for a certain number of generations right after the DFF changes. Another example is the variable local search method (Vavak et al. [1997]), in which the mutation rate is controlled using a variable local search range right after the DFF changes. Alternatively, some individuals in the population are replaced by randomly generated individuals, or simply the whole population is reinitialized randomly once the DFF changes (Hu and Eberhart [2002]).

  More recently, Woldesenbet and Yen [2009] proposed a method to adapt the population in EAs for TMO problems once the DFF changes based on historical evaluation information, in which each individual in the population right before an environmental change is relocated dimension by dimension to a new position once the DFF changes. For each dimension, the original value in the individual is moved by a distance randomly generated between

0 and a radius. For each dimension, the corresponding radius is calculated based on the corresponding fitness change due to the environmental change and the average sensitivity of an individual's position in that dimension to the corresponding change in the fitness during the evolution at last time step.

The general idea of introducing diversity right after the DFF changes is still used together with other heuristics in recent work of TMO (Parrott and Li [2006], Blackwell et al. [2008], Nguyen and Yao [2009a], Yang and Li [2010], Li and Yang [2012]).

- *Maintaining diversity all the time*: Diversity in the population is maintained at a relatively high level all the time in the hope that whenever a change happens there exist some individuals that are near a new optimum. Usually, approaches in this category do not explicitly detect environmental changes. Typical examples include the random immigrants approach (Grefenstette [1992]), the thermo-dynamical genetic algorithm (Mori et al. [1998]), the population-based incremental learning algorithm (Yang and Yao [2005]), the charged Particle Swarm Optimizer (PSO) (Blackwell and Branke [2006], Blackwell et al. [2008]), and the multi-objective approach (Bui et al. [2008]).

  In the random immigrants approach (Grefenstette [1992]), a number of individuals are replaced with randomly generated ones at every generation in the hope that the diversity of the population is maintained at a high level.

  The population in the thermo-dynamical genetic algorithm (Mori et al. [1998]) is evolved based on a metric called 'free energy'. The 'free energy' metric is the sum of the average fitness of the population and a measure of population diversity.

  Yang and Yao [2005] investigated the performance of population-based incremental learning algorithms for TMO problems, and a dual population-based incremental learning algorithm is proposed. In the dual population-based incremental learning algorithm, two probability vectors that are dual to each other are used to generate individuals at every generation. Besides, the probability vectors are updated using best found solutions at every generation. The population diversity is maintained by introducing a central probability vector that has probability 0.5 for all its entries.

Blackwell and Branke [2006] developed a multi-swarm PSO for TMO problems, in which multiple sub-swarms interact with each other both locally by an exclusion mechanism and globally by an anti-convergence mechanism, both of which aim to maintain the diversity of the swarm. Within each sub-swarm, diversity is also maintained by using charged particles or quantum particles.

Multi-objective EAs are used for solving TMO problems in (Bui et al. [2008]). A second objective, aside from the objective of TMO, is considered, the aim of which is to maintain diversity in the population. Several forms of the second objective were tested, the best of which is shown to be competitive to other TMO approaches in (Bui et al. [2008]).

- *Memory-based approaches*: Memories are used to store previously good solutions or useful information in the hope that these solutions or the information can be useful in the future. Representative methods can be divided into *implicit memory* approaches and *explicit memory* approaches. By far, most memory-based approaches belong to the category of *explicit memory* approaches.

  In *implicit memory* approaches, redundant representations for solutions are usually employed. Goldberg and Smith [1987] developed a genetic algorithm with a diploid representation and dominance operators for solving TMO problems. A diploid representation means that the chromosome of an individual has two alleles at each locus, and only one allele is dominant when the individual is being evaluated. The dominance operators adjust the dominance level of each allele as the environment changes. It was demonstrated that a genetic algorithm with the diploid representation and dominance operators is able to adapt more quickly to environmental changes than a haploid genetic algorithm or a diploid genetic algorithm with fixed dominance level. Other examples using diploid or multiploid presentations can be found in (Ng and Wong [1995], Lewis et al. [1998], Uyar and Harmanci [2005], Yang [2006b]).

In *explicit memory* approaches, an explicit memory is maintained separately from EAs and is used to bias the evolution of individuals in the population. *Explicit memory* approaches mainly differ in the following three aspects:

- the structure of the memory: Memories are in the forms of either previously found good solutions (Mori et al. [1998], Branke [1999], Bendtsen and Krink [2002], Yang [2005], Mavrovouniotis and Yang [2011]) or associative information in various forms (Eggermont et al. [2001], Yang [2006a], Simões and Costa [2008], Yang and Yao [2008], Richter and Yang [2008], Richter [2010]). Associative information is abstracted from solutions that have been evaluated by EAs over time. In terms of associative memories, Eggermont et al. [2001] stored environmental information of previous time steps. The information about the distribution of population at previous time steps are archived in (Yang [2006a]). A list of environmental states and their transitional probabilities are stored in Markov chains in (Simões and Costa [2008]). Yang and Yao [2008] developed a population-based incremental learning algorithm for TMO, in which the probability vector that generated the best solutions at previous time steps are stored. Richter and Yang [2008] memorized the probability of different regions in the solution space having good solutions. In case of dynamic constrained TMO problems, Richter [2010] stored the probability of different regions in the solution space being feasible.
- how to update the memory: There are mainly three issues with regard to updating the memory: what information is inserted to the memory; what information is deleted from the memory; and when these operations happen. In most cases, insertion into and deletion from the memory happen at every generation or right after an environmental change. In terms of what information is inserted to the memory, usually the best found solution at each generation is directly archived into the memory or used to generate the corresponding information, which is then inserted to the memory. *Explicit memory* approaches mainly

differ in their ways of deleting information from the memory: the most outdated solutions are replaced (Trojanowski and Michalewicz [1999], Simoes and Costa [2007], Woldesenbet and Yen [2009]); solutions that contribute to the population diversity least are replaced (Branke [1999], Yang [2005], Yang and Yao [2008]); solutions with worst fitness are replaced (Eggermont et al. [2001]);

  – how to use the memory: This is the issue about utilizing the memory to aid EAs in TMO. If previously found solutions are stored in the memory, usually every generation or several generations solutions in the memory are re-evaluated, and the best one is inserted to the population in EAs. If associative information is stored in the memory, the corresponding information is used to generate new solutions that are inserted into the population in EAs right after an environmental change or every fixed number of generations.

- *Multi-population approaches*: Several sub-populations are maintained to cover different promising areas in the solution space. It is hoped that once the environment changes a new optimum would be close to one of the sub-populations. The major theme in multi-population approaches is the balance between exploitation and exploration: sub-populations should track promising areas found previously and keep looking for new promising areas when the environment changes. Examples in this category include the self-organizing scouts (Branke et al. [2000]), the multi-national genetic algorithm (Ursem [2000]), the species-based PSO (Parrott and Li [2006]), and the clustering PSO (Yang and Li [2010]).

  The method of using self-organizing scouts is developed by Branke et al. [2000], in which a larger population is assigned the task of searching for new promising areas while several smaller populations, i.e., scouts, are used to track previously found promising areas. Whenever a new promising area is found, a new scout is created for that promising area, and the larger population keeps searching.

  In contrast to the self-organizing scouts (Branke et al. [2000]), a sub-population in the multi-national genetic algorithm (Ursem [2000]) is used for both tracking a previously found best solution and looking for new optimal solutions. Whenever a new local optimal solution is found by a sub-population, the sub-population is divided into two new sub-populations ensuring that a local optimum is tracked only by one sub-population in the multi-national genetic algorithm (Ursem [2000]).

In the species-based PSO (Parrott and Li [2006]), the swarm population is divided into several sub-populations, i.e., species, according to the distance between particles in an on-line manner. To be more specific, a particle called the species seed is identified for each sub-population at every iteration. The species seed is employed as the neighbourhood best for the corresponding sub-population, and particles in that sub-population evolve based on the attraction to their own personal bests and the species seed. Besides, this method was tested for both locating multiple optimum in a static environment and tracking multiple optimum in a dynamic environment.

A hierarchical clustering method is used to locate and track multiple optimum in a dynamic environment in (Yang and Li [2010]). By specifying the maximal number of sub-populations in the clustering method, a proper number of sub-populations are identified right after the environment changes. After that, identified sub-populations undergo their own local search to find promising solutions till the next environmental change. Also, previously found best solutions in each sub-population are achieved and are used to initialize a new population whenever the environment changes.

- *Prediction-based approaches*: Implicit and explicit learning models are built to predict the future based on historical evaluation information, and various types of predictions have been made to aid EAs in TMO (Branke and Mattfeld [2000], Van Hemert et al. [2001], Hatzakis and Wallace [2006], Zhou et al. [2007], Rossi et al. [2008], Simões and Costa [2009]).

  Branke and Mattfeld [2000] investigated the dynamic job shop scheduling problem using an EA with an anticipation mechanism. To be more specific, the anticipation mechanism is a combined fitness function of a schedule's idle times and tardiness. A schedule's adaptability, i.e., the idle times, to the arrival of future jobs is explicitly considered. Experimental results demonstrated the benefits of the anticipation mechanism, and the overall performance of the system was significantly improved. However, no explicit prediction model is constructed in (Branke and Mattfeld [2000]).

Van Hemert et al. [2001] introduced an EA that uses two populations to optimize the current fitness function. The first population evolves based on the current fitness function while the second population evolves based on a predicted future fitness function. Every now and then, some individuals are migrated from the second population to the first one in order that the EA is prepared for the future environmental change. However, how to predict the future fitness function is not discussed, and the experimental studies are based on a true future fitness function.

In (Hatzakis and Wallace [2006]), an Auto-regression (AR) model is built on the time series data of previously found best solutions at each previous time step. Whenever the environment changes, the trained AR model is used to predict the position of a new optimum for the new environment, and the predicted optimum is inserted into the initial population together with some individuals from the population right before the change and some randomly generated ones.

The position of an optimal solution for the next time step is predicted using a Kalman filter in (Rossi et al. [2008]). Genetic operators, i.e., mutation and crossover, and a solution's fitness are modified based on the predicted optimal solution. The advantage of the method was demonstrated in a vision-based tracking robotic application. It should be noted that the dynamic of how an optimal solution changes over time should be available to the algorithm, and this information is passed to the Kalman filter in (Rossi et al. [2008]).

Another example can be found in (Simões and Costa [2009]), where two separate learning models, i.e., a Markov chain and a non-linear regression model, are built on historical evaluation information. The Markov chain is used to predict which state the environment will change into, and the non-linear regression model aims to predict when the next change will happen. Given the output of the two learning models, the EA in (Simões and Costa

[2009]) was shown to be better prepared for the next environmental change than an EA without the two learning models. However, it should be noted that the EA in (Simões and Costa [2009]) handles an environmental change a number of generations before the change happens.

We can see from the above review that TMO approaches implicitly assume that every time the DFF changes a new solution has to be determined and implemented. This is different from ROOT as a solution's future fitness is considered in ROOT. In other words, solutions to ROOT problems are aimed to be used for multiple time steps, while solutions to TMO problems are aimed just for the current environment. It should be noted that even though there are some prediction-based TMO approaches, their purposes are TMO rather than ROOT, and more importantly they cannot be used to predict a solution's future fitness under our assumptions, which will be made clear in Section 3.4, in ROOT.

2.2 Adaptation and Optimisation

2.2.1 Problem Definition of Adaptation and Optimisation

At every time step, the goal of AO is to find a solution that maximizes the current fitness function and minimizes the adaptation cost. By adaptation cost, we mean the cost incurred when the DFF changes and an already implemented solution is adapted for the new fitness function. The adaptation cost could be measured, for instance, in the Euclidean distance between the already implemented solution and the new one. Therefore, given a DFF represented as a sequence of static fitness functions $(f_1, f_2, ..., f_N)$ during a considered time interval $[t_0, t_{end})$, a AO problem is intrinsically a DMOP and can be defined as:

$$\begin{gathered}\max[\sum_{i=1}^{N} f_i(\mathbf{x}_i), \sum_{i=1}^{N-1} -c(\mathbf{x}_i, \mathbf{x}_{i+1})] \\ s.t.\ \mathbf{x}_i\ is\ feasible,\ 1 \leq i \leq N,\end{gathered} \tag{2.2}$$

where $\mathbf{x}_i$ denotes the solution determined at time step i when the DFF takes the form of f_i. $f_i(\mathbf{x}_i)$ is the fitness of solution $\mathbf{x}_i$ at time step i, and $c(\mathbf{x}_i, \mathbf{x}_{i+1})$

denotes the adaptation cost from solution $\mathbf{x}_i$ to solution $\mathbf{x}_{i+1}$.

An implicit assumption in AO is that solving the DMOP defined in Equation 2.2 is equivalent to solving a multi-objective problem at each time step, which is, for example at time step t, maximizing f_t and $-c(\mathbf{x}_{t-1}, \mathbf{x}_t)$. Another implicit assumption in AO is that, every time the DFF changes from one fitness function to the next, a new solution has to be implemented. Also, any AO method is supposed to solve a AO problem in an on-line manner, i.e., determining a solution within a computational budget Δe once the DFF changes. Besides, the computational budget Δe should be generally smaller than the time interval between two successive environmental changes.

2.2.2 EAs for Adaptation and Optimisation Problems

The objective in AO cares not only about a solution's fitness for the current fitness function but also the adaptation cost of the current solution from the previously implemented solution (Branke [2002], Bui et al. [2012]). It is natural to think AO as a bi-objective optimisation problem at each time step as defined in Equation 2.2, where the fitness of the current solution should be maximized whereas the adaptation cost should be minimized. Nonetheless, there is some work that viewed it as a single-objective optimisation problem at each time step:

- *Single-objective approaches*: A pre-determined baseline solution is used as a starting point at each time step (Van de Vonder et al. [2007]). Whenever the DFF changes, a local search is conducted around the baseline solution, so that the resulted solution does not deviate too much from the baseline solution. However, the setting of the baseline solution is rather heuristic and can have a huge impact on the fitness of the resulted solution at each time step. In other approaches, for instance in (Van de Vonder et al. [2008]), no baseline solution is involved but a proactive solution is determined with regard to future environmental changes. In other words, a proactive solution is meant to be used for a number of consecutive time steps and therefore saves the adaptation cost of implementing a new solution whenever the DFF changes. However, future environmental changes are assumed to have a certain probabilistic distribution, and the proactive solution is achieved via Monte Carlo integration, i.e., by sampling the probabilistic distribution.

- *Bi-objective approaches*: At each time step, a solution's deviation from the previously implemented solution and its current fitness are optimized simultaneously using multi-objective EAs (Branke [2002], Bui et al. [2012]). Branke [2002] developed a multi-objective EA with the ability to explore the preferred region of the Pareto front more than other regions. This feature has its advantage over traditional multi-objective EAs in AO since in reality the current solution is required to stay near the previously implemented solution due to various constraints. Bui et al. [2012] proposed a centroid-based multi-objective EA. Basically, a set of non-dominated solutions are obtained during each time step. The centroid of the set at each time step is then calculated. Whenever the environment changes, previous centroids are exploited to give the prediction of the current centroid. The predicted centroid together with the population from the last time step are used to seed an initial population for the current environment. A case study was then conducted in military mission planning, which demonstrated the effectiveness of the centroid-based multi-objective EA.

From the above review, we can see that a solution's future fitness is not considered in AO, and it is therefore implicitly assumed in AO that every time the DFF changes a new solution has to be determined and implemented. Besides, AO offers one way to account for the adaptation cost, i.e., by requiring the current solution to be somewhat similar to the previously implemented solution. This requirement is very reactive as the previously implemented solution is determined without anticipation of how the DFF will change. In contrast, by explicitly accounting for a solution's future fitness in ROOT, solutions to ROOT problems are aimed to be used for multiple time steps. Yet, whether ROOT would strike a better balance than AO between adaptation cost and fitness over time is still an open question, which is left as our future work.

2.3 DOPs with Time-linkage

2.3.1 Problem Definition of DOPs with Time-linkage

DOPs with Time-linkage were first investigated by Bosman [2005]. The term time-linkage, first employed by Bosman [2005], is used to describe a property presented in some real world DOPs. A DOP is said to have time-linkage if decisions implemented previously for such a DOP have an influence on how the DOP changes in the future. A typical example described by Bosman [2005] is the case of dynamic vehicle routing problems, where the locations to visit are announced to the decision maker in an on-line manner. If the announced locations oscillate between two separated clusters over time, a routing decision that only considers the current location will likely lead to the oscillation of the vehicles. Yet, a more efficient routing scheme would be taking future announced locations (this may be achieved by prediction) into consideration and routing the vehicle inside one cluster first and then the other. Strictly speaking, time-linkage itself describes a property a DOP can have and does not place a requirement on how the decision maker should provide solutions to a DOP with time-linkage. In other words, the decision maker can provide solutions to a DOP with time-linkage like the way of TMO or AO. Nonetheless, existing studies in EDO on DOPs with Time-linkage (Bosman [2005], Bosman and La Poutre [2007], Nguyen and Yao [2009b], Nguyen et al. [2012b], Nguyen and Yao [2013]) implicitly assume that a new solution is determined and implemented every time the DFF changes.

Given a DFF that is represented as a sequence of static fitness functions (f_1, $f_2, ..., f_N$) during a considered time interval $[t_0, t_{end})$, the corresponding DOP with time-linkage can be defined as follows, assuming that the decision maker determines and implements a new solution at each time step and that the adaptation cost is not considered:

$$\max \sum_{i=1}^{N} f_i(\mathbf{x}_i), \tag{2.3}$$
$$s.t.\ \mathbf{x}_i\ is\ feasible,\ 1 \le i \le N,$$

where Equation 2.3 has the same form and notations as Equation 2.1 except that in Equation 2.3 solutions implemented previously have an influence on what future fitness functions look like, i.e., x_i can influence fitness function f_j, $i < j$. The exact influence is problem dependent and is usually not available to the decision maker. In other words, the decision maker has to learn the actual dependence while solving a DOP with time-linkage.

2.3.2 EAs for DOPs with Time-linkage

The goal of DOPs with time-linkage is to find an optimal solution trajectory over time instead of an optimal solution at each time step. The reason is that finding a solution that merely maximizes the current fitness function may lead to unfavourable future fitness functions. In order to solve DOPs with time-linkage optimally, prediction techniques (Bosman [2005], Bosman and La Poutre [2007], Nguyen and Yao [2009b], Nguyen et al. [2012b], Nguyen and Yao [2013]) were combined with EAs.

Bosman [2005] demonstrated that it can be arbitrarily bad just optimizing the current fitness function for a DOP with time-linkage. Therefore, Bosman [2005] claimed that it is necessary to optimize the current and the future simultaneously, based on which an algorithm framework that consists of an EA and a learning model was developed. More specially, the current solution and a future solution are optimized together. Since the future solution's fitness is evaluated by a future fitness function, a learning model is used to predict the future solution's fitness. A linear learning model and a quadratic learning model were tested, which showed that the accuracy of the prediction has a huge influence on the algorithm framework's performance. However, it should be noted that the dynamic of DOP investigated in (Bosman [2005]) is assumed deterministic. Also, in the benchmark problems used in (Bosman [2005]), a solution determined at a time step can only influence the future fitness function a certain number of time steps ahead, which may not be true in many real world cases. Besides, the algorithm framework is given the information of how many time steps ahead the current solution has an impact on. Later, Bosman and La Poutre [2007] extended the work in (Bosman [2005]) to DOPs with time-linkage where the DFFs change stochastically.

Nguyen and Yao [2009b] proposed a detailed definition framework for DOPs with time-linkage, where the static form of fitness function, the inherent dynamic of fitness function, and the time-linkage property can be specified separately. Also, a new type of DOPs with time-linkage was identified, in which the dynamic of the DOPs is unpredictable to some extent. Nguyen and Yao [2009b] termed the new property as prediction-deceptive. For this new type of DOPs with time-linkage and prediction-deceptive, Nguyen and Yao [2009b] showed that an EA with existing prediction methods can perform even worse than an EA without prediction. Finally, a new benchmark problem was developed to study the new type of DOPs with time-linkage and prediction-deceptive. Later, Nguyen et al. [2012b] developed a specific approach for the solving of DOPs with time-linkage and prediction-deceptive under certain circumstances. More specially, the prediction-deceptive is assumed to be the case where the time-linkage property takes several function forms and is switching among those forms over time. Moreover, the switch rules are assumed to be known to the decision maker. Two new benchmark problems for DOPs with time-linkage and prediction-deceptive were also developed in (Nguyen et al. [2012b]).

It should be noted that existing studies (Bosman [2005], Bosman and La Poutre [2007], Nguyen and Yao [2009b], Nguyen et al. [2012b], Nguyen and Yao [2013]) on DOPs with time-linkage implicitly assume that a new solution is implemented every time the DFF changes, whereas in ROOT an implemented solution is used for multiple consecutive time steps. We can also incorporate time-linkage into ROOT problems, although in this thesis we will focus on ROOT problems without time-linkage.

2.4 ROOT and Similar Motivations

The work on ROOT has started since a few years ago. The idea of ROOT was initially proposed by Yu et al. [2010], where it is suggested to find robust solutions to DOPs based on the investigation that it is unrealistic to TMO in many real world DOPs. Yet, a clear and practical definition of ROOT is lacking in (Yu et al. [2010]), except for some evaluation criteria for comparing an algorithm's performance on ROOT. Also, the robustness of solutions to ROOT problems is not explicitly defined in (Yu et al. [2010]), without of which it may be difficult to evaluate a ROOT solution.

We gave a definition of ROOT in (Fu et al. [2012]), based on which we developed a number of measures that can be used to characterize and analyse the dynamic of the DFF when solving ROOT problems. Those measures can be used to understand what aspects of a DFF change and more importantly how these changes influence the solving of ROOT problems. Yet, the problem definition of ROOT in (Fu et al. [2012]) is of little relevance to real world DOPs where a robust solution is desirable. The reasons are two-fold. Firstly, within the problem definition of ROOT in (Fu et al. [2012]), the objective of the decision maker is to find solutions whose performance does not deviate too much from true optimal solutions. However, often in practice, the information of true optimal solutions is not available. Secondly, a clear robustness definition of solutions to ROOT problems is still lacking in (Fu et al. [2012]).

Jin et al. [2012] developed an algorithm framework for ROOT, in which a solution is evaluated in the algorithm framework by the average of its past, current, and future fitness. To be more specific, a solution's past fitness is approximated by building a surrogate model using neighbourhood solutions evaluated at the same previous time step, while the solution's current fitness is evaluated using the current fitness function. By modelling a solution's past and current fitness as a time series, the solution's future fitness is predicted by training a learning model on the time series. The algorithm framework was then tested on several benchmark problems, which demonstrated its effectiveness on three performance measures. Yet, the three performance measures are not directly connected to a solution's robustness definition in (Jin et al. [2012]).

By pointing out the lack or inappropriateness of robustness definition in ROOT in (Yu et al. [2010], Fu et al. [2012], Jin et al. [2012]), we proposed two new robustness definitions in (Fu et al. [2013]). Also, we extended the algorithm framework proposed by Jin et al. [2012] with two new metrics for respectively maximizing the two proposed robustness definitions in (Fu et al. [2013]).

The motivation of ROOT is to find robust solutions that can be used for multiple time steps during which the DFF changes, and there have been similar motivations in the literature (Handa [2006], Leung et al. [2007], Van de Vonder et al. [2008], Salomon et al. [2013]). Robust solutions to DFFs, in which a robust solution is used for multiple time steps, are sought in (Handa [2006], Van de Vonder et al. [2008]). However, future fitness functions are assumed either to be known (Handa [2006]) or to follow a certain probabilistic distribution (Van de Vonder et al. [2008]), neither of which is assumed in ROOT. Leung et al. [2007] developed a robust optimisation model for a multi-site aggregate production planning problem, where a solution's fitness and feasibility are subject to environmental uncertainties. A robust solution is optimal if both its fitness and feasibility remain "close" to optimality under any environmental change. However, a fixed probability distribution of the environmental uncertainties is assumed in (Leung et al. [2007]), which is not assumed in ROOT. Salomon et al. [2013] addressed the active robust optimisation problem, where a solution contains some adjustable parameters that can be adapted while in use. An optimal robust solution in this case is a solution that maximizes its performance under different environments and minimizes the corresponding cost incurred. The cost comes from three components, namely the implementation cost of the solution, the operational cost of maintaining a state of the adjustable parameters in a particular situation, and the adaptation cost of adapting the adjustable parameters in reacting to environmental changes. Similarly, the distribution of the dynamic environment is assumed to be available in (Salomon et al. [2013]) in order to be sampled for evaluation of a solution, and this is not assumed in ROOT. To summarise similar motivations to ROOT in the literature, they assume either that the decision maker knows the future fitness function or that the future fitness function has a fixed probabilistic distribution that is available to the decision maker, both of which may be impractical in most real world DOPs and are not assumed in ROOT. In other words, ROOT presents new challenges that have not been addressed before in the EDO literature.

2.5 Differences Between ROOT and Traditional Evolutionary Robust Optimisation

Robust solutions are also sought in traditional Evolutionary Robust Optimisation (ERO) (Jin and Branke [2005], Beyer and Sendhoff [2007]). Yet, robust solutions in ERO are significantly different from robust solutions in ROOT.

In ERO, instead of finding an optimal solution maximizing a fitness function f, as defined in Definition 1.1.1, the following expected fitness is usually maximized:

$$\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(\mathbf{x}+\boldsymbol{\delta},\boldsymbol{\alpha})p(\boldsymbol{\delta})p(\boldsymbol{\alpha})d\boldsymbol{\delta}d\boldsymbol{\alpha}, \tag{2.4}$$

where $\mathbf{x}$ denotes a vector of decision variables. $\boldsymbol{\delta}$ represents noises added to $\mathbf{x}$, and $\boldsymbol{\delta}$ follows a probabilistic distribution $p(\boldsymbol{\delta})$. $\boldsymbol{\alpha}$ represents environmental states that specify f, and $\boldsymbol{\alpha}$ follows a probabilistic distribution $p(\boldsymbol{\alpha})$. Both $p(\boldsymbol{\delta})$ and $p(\boldsymbol{\alpha})$ are assumed to be available to the decision maker. The reason in introducing $\boldsymbol{\delta}$ is that in some cases implemented decision variables can be slightly different from determined decision variables due to manufacturing tolerances (Paenke et al. [2006]). The reason in modelling $\boldsymbol{\alpha}$ as a random variable is that in some cases environmental states $\boldsymbol{\alpha}$ is noisy (Beyer and Sendhoff [2007]).

From Equation 2.4, we can see that in ERO environmental states, i.e., $\boldsymbol{\alpha}$, are modelled as random variables, whose probabilistic distribution $p(\boldsymbol{\alpha})$ is assumed to be known. In contrast, $\boldsymbol{\alpha}$ in ROOT is modelled as a discrete time series[1], which results in a sequence of fitness functions over time, i.e., one fitness function at one time step. In other words, the ERO robustness in Equation 2.4 can be either mathematically calculated if a closed form of Equation 2.4 is available or otherwise estimated by Monte Carlo integration, i.e., by sampling over $\boldsymbol{\delta}$ and $\boldsymbol{\alpha}$. On the other hand, a robust solution in ROOT, which will be discussed in detail later in Chapter 3, involves a solution's future fitness. A solution's future fitness is determined by $\boldsymbol{\alpha}$ at future time steps in ROOT and can only be predicted based on previous evaluation information. Therefore, past information

[1]In this thesis, $\boldsymbol{\alpha}$ in ROOT is assumed to be deterministic at each time step.

and appropriate learning models are crucial to find robust solutions in ROOT. In other words, ERO methods can not be directly applied to ROOT problems. It should be noted that, although we discuss the differences between ERO and ROOT using the expected fitness in Equation 2.4, the same differences apply to other ERO robustness definitions (Beyer and Sendhoff [2007]) as well. To sum it up, ROOT presents new challenges that have not been addressed before in the ERO literature.

2.6 Summary

In this chapter, we have briefly reviewed different types of DOPs and the corresponding methods in EDO. More importantly, the unique feature of ROOT, which is finding robust solutions that can be used for multiple time steps, was emphasized by making a comparison to other types of DOPs. The brief literature review in EDO shows that ROOT problems present new challenges that have not been investigated before in other types of DOPs in EDO. Also, the differences between ROOT and traditional ERO were discussed. Although the unique feature of ROOT has been stated clearly, the motivation of ROOT has not been strongly validated, and a practical problem definition of ROOT is lacking. Moreover, the robustness of a ROOT solution is not explicitly defined. In the next chapter, we will motivate ROOT by investigating real world DOPs where robust solutions that can be used after environmental changes are desirable. Based on the investigation, a practical problem definition of ROOT will be given. Moreover, two robustness definitions will be defined, both of which are motivated respectively by investigating two typical situations of real world DOPs where a robust solution is favourable.

CHAPTER 3

MOTIVATION, PROBLEM DEFINITION, AND PROBLEM DIFFICULTIES OF ROBUST OPTIMISATION OVER TIME

In Chapter 2, different types of DOPs studied in EDO and their corresponding methods were reviewed. The unique feature of ROOT was identified, which is finding robust solutions that can be used for a long period during which environmental changes happen. Existing work on ROOT was also reviewed, which shows that a practical problem definition of ROOT that captures the unique feature of ROOT is lacking. Also, the robustness of a ROOT solution is not clearly defined. In this chapter, we try to solve these problems. Section 3.1 motivates ROOT by investigating real world DOP applications where it is desirable to have robust solutions that can be used after environmental changes. Based on the investigation in Section 3.1, a problem definition of ROOT is given in Section 3.2, and two robustness definitions are proposed in Section 3.3. Assumptions with regard to the ROOT problems investigated in this thesis are made clear in Section 3.4. Problem difficulties of ROOT, from the perspective of an EA, are briefly discussed in Section 3.5. Finally, a summary is given in Section 3.6.

3.1 Motivation of ROOT

For static optimisation problems defined in 1.1.1, usually only one solution, which is found to be best in terms of fitness for the corresponding objective fitness function among all evaluated solutions, is determined and implemented in practice.

However, for optimisation problems that change over time, i.e., DOPs, the decision maker has to repeatedly determine and implement solutions in reacting to environmental changes. So far, the majority of research on applying EAs to DOPs has been focusing types of DOPs where it is implicitly assumed that the decision maker has to determine and implement a new solution every time the environment changes. Implementing a new solution every time the environment changes is justifiable only in situations where implementing a solution, e.g., updating some control variables, can be finished instantaneously and cheaply, such as in (Stroud [2001], Ursem et al. [2002], Rossi et al. [2008]).

However, in situations where the implementation of a solution involves human operation (Handa et al. [2005]), resource transportation (Bui et al. [2012]), etc, which all incur a huge cost, it might be impossible to implement a new solution every time the environment changes. This is further explained in the following:

1. A lot of real world DOPs involve human operation. Taking practical dynamic vehicle routing problems for example (Larsen and Madsen [2000], Handa et al. [2005], Handa [2006]), the environmental states, e.g., demands from customers, conditions of road, etc., vary from day to day. For this kind of DOPs, if we implement a different solution every day, it will cause disruptions to the working timetable of staff and may also confuse the operators as the operators would daily change their implementations (Handa [2006]). Therefore, in such circumstances, it would be more beneficial having a fixed and good solution implemented and used for a long time period than implementing a new solution every time the environment changes.

2. In some circumstances, it is desirable to have an already implemented solution in use as much time as possible, during which environmental changes may happen, as long as the solution maintains its feasibility and its performance above a certain level. Such circumstances can be found in aircraft taking-off/landing scheduling problems (Atkin et al. [2008], Wilkins et al. [2008]) and the airspace partition problem (Gianazza [2010]). As long as the system performance is maintained above a certain level, it is preferred to stick to an already implemented solution/schedule after an environmental change. The reason is that any modification of an already implemented so-

lution will cause unfavourable disruptions to the operations in the airport, e.g., rescheduling some other aeroplanes. Therefore, in such circumstances, it is more practical to use a solution that can maintain its feasibility and its performance above a threshold as long as possible, starting from the time when it is first implemented, than to use a solution that is determined by, e.g., maximizing the performance only for the current environment.

From the above discussion, we can see that implementing a new solution every time the environment changes is not justifiable in situations where an implemented solution is required to be kept in use for a long period during which environmental changes may happen. In other words, solutions that have good performance for not only the current environment but also environments after environmental changes are desirable in those situations. We term such solutions as robust solutions in the thesis. Supposing a DFF can be represented as a sequence of fitness functions $(f_1, f_2, ..., f_N)$ during a considered time interval $[t_0, t_{end})$, a robust solution is defined as follows:

Definition 3.1.1. *Supposing the current time step is t when the fitness function is f_t, a robust solution at time step t is a solution whose performance is measured not only by f_t but also future fitness functions f_{t+i}, $i \geq 1$.*

The problem of finding a robust solution at each time step for a DFF is referred to as ROOT, and this thesis is dedicated to the research of ROOT.

3.2 A Formal Problem Definition of ROOT

As we discussed in Section 3.1, the idea of ROOT is to repeatedly find robust solutions at each time step. Every time the environment changes from one static fitness function to the next one, we need to determine a robust solution within a computational/time budget Δe. It should be noted that we may not need to actually implement the solution if the previously implemented solution is still in use for the current time step. Given a DFF that is represented as a sequence of static fitness functions $(f_1, f_2, ..., f_N)$ during a considered time interval $[t_0, t_{end})$, a ROOT problem is defined as:

$$\max \sum_{i=1}^{N} F(\mathbf{x}_i), \tag{3.1}$$
$$s.t.\ \mathbf{x}_i\ is\ feasible,\ 1 \leq i \leq N,$$

where $\mathbf{x}_i$ denotes the robust solution determined at time step i when the DFF takes the form of f_i. $F(\mathbf{x}_i)$ is the robustness of solution $\mathbf{x}_i$, and F is a user-defined function of a solution's current fitness f_i and its future fitness f_{i+j}, $j \geq 1$. It should be noted that any ROOT method is supposed to solve a ROOT problem in an on-line manner. In other words, a ROOT method starts when the DFF takes the form of f_1 and determines a solution $\mathbf{x}_1$ within the computational budget Δe. Then, the ROOT method will determine a solution $\mathbf{x}_2$ within the computational budget Δe after the DFF changes from f_1 to f_2. The process repeats till the last static fitness function f_N. Also, the computational budget Δe should be generally smaller than the time interval between two successive environmental changes in ROOT.

In the ROOT problem definition in Equation 3.1, we have not explicitly defined the robustness of a solution in ROOT, and the purpose is to make the ROOT problem definition flexible enough to incorporate different robustness definitions. Nonetheless, in this thesis, we will investigate only two robustness definitions, both of which have been proposed by investigating the characteristics of real world DOPs where robust solutions are desirable (Fu et al. [2013]). The two robustness definitions are defined in the following section.

3.3 Two Robustness Definitions: 'Average Fitness' and 'Survival Time'

In Section 3.1, we discussed two situations of DOPs where a robust solution is desirable. To account for the first situation where it is desirable to use a solution for a long time period, during which environmental changes occur, the following robustness definition is proposed to quantify the performance of a solution during a time period. Given a DFF that is represented as a sequence of fitness functions $(f_1, f_2, ..., f_N)$ during a considered time interval $[t_0, t_{end})$, assuming at time step t a solution $\mathbf{x}_t$ is determined, the first robustness definition, which is called 'average fitness', is defined as:

$$F^a(\mathbf{x}_t, t, T) = \frac{1}{T}\sum_{i=0}^{T-1} f_{t+i}(\mathbf{x}_t), \tag{3.2}$$

where T is a user-defined time window parameter, which specifies for how many consecutive time steps a solution's fitness needs to be considered.

An illustration of a solution's 'average fitness' is presented in Figure 3.1, in which a solution's fitness time series at different time steps is plotted. Without loss of generality, supposing that the solution in Figure 3.1 is determined or implemented at time step 4 and that the time window T is set to 4, the solution's 'average fitness' is the average of its fitness at time step 4, 5, 6, and 7, which is 26.5 in this case. It should be noted that, when maximizing 'average fitness' in ROOT, only a solution's fitness at the current time step can be evaluated while a solution's fitness at future time steps can only be predicted. In the case of Figure 3.1, this means only the fitness at time step 4 can be evaluated while the fitness at time step 5, 6, 7, etc. should be predicted. A more detailed discussion of assumptions with regard to ROOT problems investigated in this thesis is presented later in Section 3.4 of this chapter.

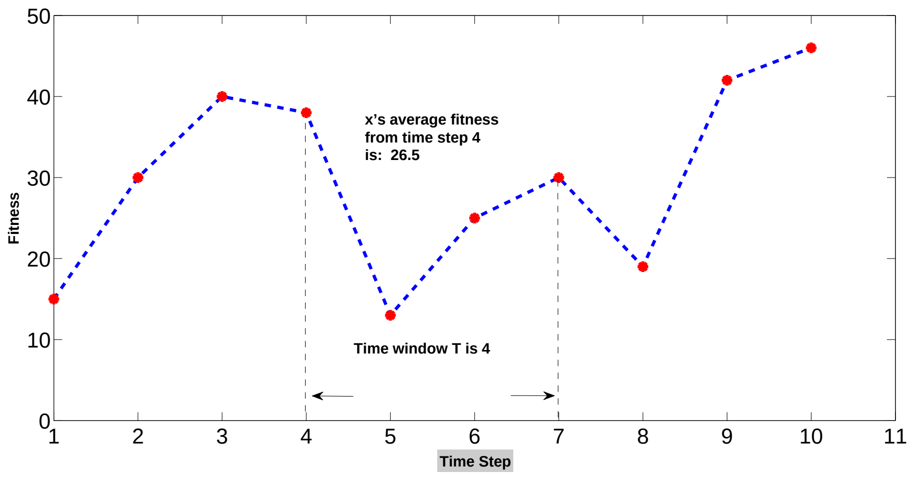


Figure 3.1: Illustration of a solution's 'average fitness'.

To account for the second situation where it is desirable to use a solution for as much time as possible, as long as the solution maintains its fitness above a predefined threshold, the following robustness definition is proposed to quantify the amount of time a solution can maintain its fitness above a user-specified threshold. Given a DFF that is represented as a sequence of fitness functions $(f_1, f_2, ..., f_N)$ during a considered time interval $[t_0, t_{end})$, assuming a solution $\mathbf{x}_t$ is determined at time step t, the second robustness definition, which is called 'survival time', is defined as:

$$F^s(\mathbf{x}_t, t, V) = \begin{cases} 0, & \text{if } f_t(\mathbf{x}_t) < V, \\ 1 + max\{l | f_i(\mathbf{x}_t) \geq V, \forall i, t \leq i \leq t + l\}, & \text{otherwise,} \end{cases} \tag{3.3}$$

where V is a user-defined fitness threshold. The robustness 'survival time' quantifies the number of consecutive time steps a solution maintains its fitness above V since the time step when it is determined.

An illustration of a solution's 'survival time' is presented in Figure 3.2, in which a solution's fitness time series at different time steps is plotted. Without loss of generality, suppose that the solution in Figure 3.2 is determined or implemented at time step 4 and that the fitness threshold V is set to a value that is plotted as a dashed horizontal line in Figure 3.2. We can see that, from Figure 3.2, the solution maintains its fitness above the fitness threshold V for 4 time steps starting from time step 4 (at time step 8, the solution's fitness drops below the fitness threshold V). As a result, according to the robustness definition of 'survival time' in Equation 3.3, the 'survival time' of the solution in Figure 3.2 is 4. Again, it should be noted that, when maximizing 'survival time' in ROOT, only a solution's fitness at the current time step can be evaluated while a solution's fitness at future time steps can only be predicted. In the case of Figure 3.2, this means only the fitness at time step 4 can be evaluated while the fitness at time step 5, 6, 7, etc. should be predicted.

A more detailed discussion of assumptions with regard to ROOT problems investigated in this thesis is presented later in Section 3.4 of this chapter.

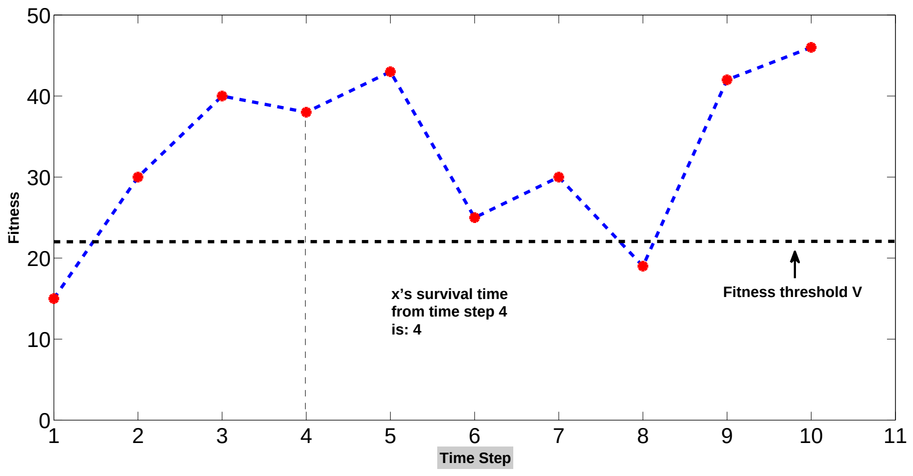


Figure 3.2: Illustration of a solution's 'survival time'.

3.4 Assumptions for ROOT Problems Investigated in This Thesis

It is important that implicit assumptions made explicitly about ROOT, so people can understand the potential applications of ROOT. So far, we have explicitly assumed that the dynamic environment in a ROOT problem can be represented as a sequence of fitness functions over time. In the following, a full list of assumptions about ROOT problems studied in this thesis is made explicit. These assumptions are established to make ROOT problems studied in this thesis practical on the one hand and easily formulated on the other hand. More importantly, these assumptions are established based on a comprehensive investigation of real world DOPs in Chapter 3 in the thesis (Nguyen [2011]).

Firstly, the dynamic environment, i.e., the DFF, is assumed to change suddenly with constant status between two successive changes. This means the DFF can be represented as a sequence of static fitness functions during a considered time interval. The primary reason to assume that the dynamic environment changes discretely is as follows. It is natural to divide those types of DOPs where implementing a solution involves human operation or huge cost into time windows, during each of which the environment is considered static.

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

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

Fourthly, a solution's future fitness[1] should be predictable to some extent. In ROOT, at the current time step, we would like to find a solution that maximizes 'average fitness' or 'survival time', both of which involve a solution's future fitness. As we are unable to evaluate a solution's future fitness at the current time step, certain prediction models are needed, which will be trained based on historical evaluation information. The historical evaluation information consists of a set of evaluated solutions with their fitnesses at previous time steps, which is essentially a set of triplets in the form $(\mathbf{x}, f_i(\mathbf{x}), i)$. $(\mathbf{x}, f_i(\mathbf{x}), i)$ means solution $\mathbf{x}$ has been evaluated at previous time step i with its fitness for f_i being $f_i(\mathbf{x})$. Only under the assumption about predictability in a solution's future fitness, information gathered in the past is useful to guide the search for the present in ROOT. Otherwise, the environment is thought to change completely randomly, and there is no need to use historical evaluation information.

Finally, solutions implemented in ROOT are hoped to be used for a long time period, during which the environment is changing. Therefore, it is required that the dimensionality of decision variables does not change over time. However, it should be noted that we can have the constraints for decision variables changing

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

over time in ROOT, although in this thesis we only investigate the case where the solution space $\mathcal{S}$ stays constant over time.

Based on these assumptions, we briefly analyse the problem difficulties of ROOT in the following section from the perspective of an EA, which is informative in designing specialised algorithms for ROOT problems.

3.5 Problem Difficulties of ROOT

The task of ROOT at each time step is to find an optimal robust solution. Bear in mind with the assumptions of ROOT listed in Section 3.4, we can see that at the current time step, in order to find an optimal robust solution in terms of 'average fitness' or 'survival time', a solution's future fitness needs to be predicted. This imposes two major difficulties in ROOT problems. The first one would be in building learning models for the prediction task. Given historical evaluation information, it is not straightforward as how to build a learning model that can be used to predict a solution's future fitness. On the other hand, there is no guarantee that a solution's future fitness can be predicted perfectly. Therefore, when we apply EAs to search for ROOT solutions, solutions are evolved based on inaccurate 'average fitness' or 'survival time'. In the following, we discuss in detail difficulties encountered in ROOT from both perspectives.

3.5.1 Difficulties in Predicting a Solution's Future Fitness in ROOT

The most intuitive way to predict a solution's future fitness would be formulating it as a time series prediction problem and then employing time series prediction models, e.g., the AR model (Akaike [1969]), to predict a solution's future fitness. However, should the time series data be a solution's past fitnesses or models of past fitness functions does not have a confirm answer. Moreover, how to generate the time series data is not straightforward. For instance, if we decide the time series data is a series of a solution's previous and current fitness, it may happen that a solution has not been evaluated at certain previous time steps, and therefore the solution's fitness at some previous time steps cannot be fetched directly from historical evaluation information. Finally, given the time series data, the time series prediction task is itself a hard problem (Box and Jenkins [1994]).

3.5.2 Difficulties in Evolving a Population Based on Inaccurate Information in ROOT

We are interested in applying EAs to searching for robust solutions to ROOT problems. The metric that is used to differentiate good solutions from bad ones in EAs plays an important role in the optimisation process, since it guides the population to converge to good solutions. In ROOT, a solution is evaluated in EAs by the solution's estimated/predicted 'average fitness' or 'survival time'. Since a solution's future fitness cannot be exactly evaluated, the estimated/predicted 'average fitness' or 'survival time' is inevitably inaccurate or noisy. Evolving solutions based on an inaccurate or noisy metric has long been studied in evolutionary optimisation with noisy fitness functions (Jin and Branke [2005]) and surrogate-assisted evolutionary optimisation (Jin [2005]). A major concern with an inaccurate or noisy metric is that it may lead EAs to converge to a false optimum. This concern applies to EAs for ROOT problems as well.

3.6 Summary

In this chapter, we have motivated ROOT by investigating real world DOPs in two situations. In the first situation, a solution that has good performance over a time period during which environmental changes can occur is sought. In the second situation, a solution is kept in use as long as its performance is maintained above a certain threshold. The first contribution of this chapter is a practical problem definition of ROOT, in which the dynamic environment is assumed to be represented as a sequence of fitness functions over time. The task of ROOT at each time step is then finding a solution that maximizes a corresponding robustness definition.

The second contribution of this chapter is the two robustness definitions proposed to respectively account for the two situations that motivate ROOT. One robustness definition is ‘average fitness’, and the other is ‘survival time’. Assumptions for ROOT problems studied in this thesis were then made explicitly, so people can understand the potential applications of ROOT. Moreover, we briefly analysed the problem difficulties of ROOT from the perspective of an EA.

A single objective, namely the robustness of a solution (i.e., ‘average fitness’ or ‘survival time’), is considered at a time step currently in ROOT. For the future work, it would then be interesting to extend the current definition of ROOT to multi-objective cases as many real world DOPs are multi-objective in nature. A practical second objective would be the adaptation cost from a previously implemented solution to a new implemented solution, which has been considered in AO problems. Moreover, it is worth exploring other robustness definitions that are relevant to practical DOP applications, given that a solution’s robustness involves its fitness after environmental changes.

CHAPTER 4

BENCHMARKING ROBUST OPTIMISATION OVER TIME

In Chapter 3, we gave a practical problem definition of ROOT in Section 3.2, where the task of ROOT is to find a robust solution within a computational budget at each time step. Two robustness definitions, i.e., ‘average fitness’ and ‘survival time’, that reflect real world DOPs characteristics were also defined in Section 3.3. Before an algorithm is applied to real world DOPs, it is good to test it on benchmark problems as potential advantages and disadvantages of the algorithm can be understood. This chapter discusses the benchmark design in terms of maximizing ‘average fitness’ and ‘survival time’ in ROOT, which can be seen as a prior step to developing algorithms for ROOT problems. Section 4.1 briefly reviews existing DOP benchmarks and discusses the problems when they are used as ROOT benchmarks. We then develop two ROOT benchmark problems to alleviate these problems in Section 4.2. One ROOT benchmark problem is used for maximizing ‘average fitness’ in ROOT, and the other is used for maximizing ‘survival time’ in ROOT. ROOT methods from the literature are then tested on the two ROOT benchmarks in Section 4.3. Finally, Section 4.4 summaries this chapter.

4.1 Existing DOP Benchmarks and Their Problems for ROOT

In order to evaluate algorithms for ROOT, appropriate benchmarks are needed. Over the years, researchers have developed various DOP benchmarks to test an algorithm's TMO ability. Since existing DOP benchmarks are essentially benchmarks that define the DFFs of DOPs, those DOP benchmarks could be potentially used for testing an algorithm's ROOT ability as well.

There are mainly three categories in existing DOP/DFF benchmarks. In the first category, the DFF switches among several fixed fitness functions, e.g., the oscillating fitness landscape (Cobb [1990]) and the dynamic knapsack problem (Lewis et al. [1998]).

The second category includes benchmarks that are built by defining baseline fitness functions with configurable parameters and dynamics to change those configurable parameters. Typical examples can be found in the moving peaks benchmark (Branke [1999]), DF1 dynamic benchmark (Morrison and De Jong [1999]), the multi-objective dynamic test problem generator (Jin and Sendhoff [2004]), the dynamic rotation peak benchmark together with the dynamic composition benchmark (Li et al. [2008]), and the dynamic constrained benchmark (Nguyen and Yao [2009a]).

Compared with the first two categories, in which the DFF changes over time, the third category involves benchmarks where the DFF stays unchanged but a solution $\mathbf{x}$ has to go through a transformation before being evaluated and the transformation rule is subject to environmental changes. A representative example is the exclusive-or (XOR) generator for binary encoded problems (Yang [2003]) and continuous domains (Tinós and Yang [2007]). In contrast to the fitness-landscape-oriented DOP benchmarks in the aforementioned three categories, another seminal work was developed in (Ursem et al. [2002]), where the authors proposed a DOP benchmark based on general characteristics of real world DOPs.

Since so many DOP benchmarks are available, an immediate question is whether any of these could be employed directly to serve as a ROOT benchmark.

It turns out that it is not sufficient to use existing DOP benchmarks as ROOT benchmarks when maximizing 'average fitness' or 'survival time'. The reason is that it is generally difficult to know the absolute best 'average fitness' for existing DOP benchmarks given a time window T ($T > 2$). Also, it is generally difficult to know the absolute best 'survival time' for existing DOP benchmarks given a fitness threshold V.

Given the 'average fitness' definition in Equation 3.2 in Section 3.3, the absolute best 'average fitness' starting from time step t with time window T, $t+T \leq N$, is:

$$\max\{F^a(\mathbf{x}, t, T) | \mathbf{x} \in \mathcal{S}\}, \tag{4.1}$$

where $\mathcal{S}$ denotes the solution space of $\mathbf{x}$. Given the 'survival time' definition in Equation 3.3 in Section 3.3, the absolute best 'survival time' starting from time step t with the fitness threshold V is:

$$\max\{F^s(\mathbf{x}, t, V) | \mathbf{x} \in \mathcal{S}\}. \tag{4.2}$$

The lack of knowledge about the absolute best performance makes the evaluation and comparison of algorithms on existing benchmarks difficult if not impossible. Therefore, it is not sufficient to use existing DOP benchmarks for the study of ROOT. The reasons why it is so difficult to know the absolute best 'average fitness' or the absolute best 'survival time' are explained respectively in the following subsections.

4.1.1 Difficulties in Calculating the Absolute Best 'Average Fitness'

The difficulty to obtain the absolute best 'average fitness' for existing DFF benchmarks lies in the fact that usually the function form of $F^a(\mathbf{x}, t, T)$ when $T \geq 2$ cannot be reduced to the same form as the corresponding baseline fitness function. Therefore, the logical inference procedure that is used to obtain an optimum of the corresponding baseline fitness function cannot be used to obtain an optimum of the 'average fitness' function in Equation 3.2 in Section 3.3.

Taking the widely used moving peaks benchmark (Branke [1999]) and the DF1 dynamic benchmark (Morrison and De Jong [1999]) for example, the baseline fitness function in both benchmarks takes a similar form as follows:

$$f_t(\mathbf{x}) = \max_{i=1}^{i=m}\{h_t^i - w_t^i * ||\mathbf{x} - \mathbf{c}_t^i||_2\}, \tag{4.3}$$

where scalars h_t^i, w_t^i, and vector $\mathbf{c}_t^i, \mathbf{c}_t^i = (c_1^i, ..., c_d^i)$, denote the height, the width and the centre of the ith peak function at time step t; $\mathbf{x}$ is the vector of decision variables; m is the total number of peaks, and d is the number of dimensions. According to the function form in Equation 4.3 and supposing the solution space $\mathcal{S}$ being the d-dimensional vector space over the real numbers $\mathcal{R}^d$, it is easy to infer that an optimal solution for the baseline fitness function in Equation 4.3 is $\mathbf{c}_t^{j^*}$ with its fitness taking the value $h_t^{j^*}$ where $j^* = \arg\max_i h_t^i, 1 \leq i \leq m$. When we use the baseline fitness function in Equation 4.3 to generate the DFF, the function form of 'average fitness', $F^a(\mathbf{x}, t, T)$, becomes (supposing $T = 2$):

$$\frac{1}{2}(\max_{i=1}^{i=m}\{h_t^i - w_t^i * ||\mathbf{x} - \mathbf{c}_t^i||_2\} + \max_{i=1}^{i=m}\{h_{t+1}^i - w_{t+1}^i * ||\mathbf{x} - \mathbf{c}_{t+1}^i||_2\}). \tag{4.4}$$

It is easy to verify that we are unable to use the same inference technique to obtain the absolute best 'average fitness' ($T \geq 2$). The reason is that the function form of 'average fitness' in Equation 4.4 cannot be reduced to the same form as the baseline fitness function in Equation 4.3.

A natural attempt to find the absolute best 'average fitness' for DFFs generated using the baseline fitness function in Equation 4.3 would be to set the derivative of the function of 'average fitness' to zero and then obtain the stationary points. However, the analytical solutions of stationary points are not available because of the function form in Equation 4.4.

Although we illustrate the difficulty in calculating the absolute best 'average fitness' using two existing DFF benchmarks, we argue that for most existing DFF benchmarks (Branke [1999], Morrison and De Jong [1999], Li et al. [2008]), the same difficulty in calculating the absolute best 'average fitness' applies. However, there do exists one DFF benchmark (Jin and Sendhoff [2004]) where the function form of 'average fitness' can be reduced to the form of the corresponding baseline fitness function, and therefore it is capable to calculate the absolute best 'average fitness' just as calculating the optimal fitness for the baseline fitness function in (Jin and Sendhoff [2004]). Nonetheless, we would like to contribute a new benchmark for maximizing 'average fitness' in ROOT. This is partly because, for the DFF benchmark in (Jin and Sendhoff [2004]), all possible optimal solutions of 'average fitness' are confined within a fixed Pareto set. The reason is that the DFF is generated in (Jin and Sendhoff [2004]) by aggregating different objects of a fixed multi-objective optimisation problem and changing the aggregating weights.

4.1.2 Difficulties in Calculating the Absolute Best 'Survival Time'

The difficulty in calculating the absolute best 'survival time' in Equation 4.2 originates from the difficulty associated with general non-linear arithmetic Constraint Satisfaction Problems (CSPs) (Ratschan [2006]). Given a sequence of static fitness functions $(f_1, f_2, ..., f_N)$, the problem of calculating the absolute best 'survival time' starting from f_t is equivalent to the problem of finding the maximal number that the variable l can take for which there exists a solution $\mathbf{x}^*$, $\mathbf{x}^* \in \mathcal{S}$, satisfying the following array of arithmetic constraints simultaneously:

$$\begin{cases} f_t(\mathbf{x}^*) \geq V, \\ f_{t+1}(\mathbf{x}^*) \geq V, \\ \vdots \\ f_{t+l}(\mathbf{x}^*) \geq V. \end{cases} \tag{4.5}$$

The problem of answering whether there exists a solution $\mathbf{x}^* \in \mathcal{S}$ that satisfies all the constraints in Equation 4.5 for a fixed l, $l \geq 0$, is one type of CSPs. Therefore, being able to solve the corresponding CSP is a necessary condition of calculating the absolute best 'survival time'. Suppose we can solve the corresponding CSP successfully, we can obtain the absolute best 'survival time', starting from any fitness function f_t, $1 \leq t \leq N$, in the sequence $(f_1, f_2, ..., f_N)$, by separately solving a set of corresponding CSPs with l being $1, 2, ..., N - t$.

As stated in (Ratschan [2006]), solving arbitrary non-linear arithmetic CSPs over the real numbers is undecidable. This means if f_{t+i}, $0 \leq i \leq l$, is an arbitrary non-linear function of $\mathbf{x}$, it is impossible to construct a single algorithm that will always lead to a yes/no answer as to whether there exists a solution $\mathbf{x}^*$ simultaneously satisfying all the constraints in Equation 4.5. If f_{t+i}, $0 \leq i \leq l$, is a linear function of $\mathbf{x}$, the corresponding CSP in Equation 4.5 is reduced to a linear programming problem and can be solved in polynomial time. Also, if both the baseline fitness function f_{t+i}, $0 \leq i \leq l$, and the solution space $\mathcal{S}$ are convex, the corresponding CSP can be solved satisfactorily (Nuzzo et al. [2010]).

Based on the above discussions, we can see that in order to employ a general CSP solver to solve the CSP with a fixed l in Equation 4.5, the baseline fitness function f_t is required to be either linear or convex together with a convex solution space $\mathcal{S}$. Alternatively, we can require the baseline fitness function to take a specific form (more complexed than just linear or convex) based on which we can calculate the absolute best 'survival time'. We take the latter way in this chapter to develop a new baseline fitness function (i.e., a new baseline fitness function for a benchmark) for the study of maximizing 'survival time' in ROOT. The reason is that the requirement of the baseline fitness function being linear or convex is too strong, and it may only represent a small portion of real world problems.

4.2 Tunable Benchmark Problems for ROOT

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

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

Since the two proposed ROOT benchmarks have the 'peak' characteristic as defined in the moving peaks benchmark (Branke [1999]) and aim at testing an algorithm's ROOT ability (in terms of maximizing 'average fitness' and 'survival time' respectively), we term the two benchmarks Robust Moving Peaks Benchmark (RMPB). The benchmark for maximizing 'average fitness' is denoted as RMPB-I, and the benchmark for maximizing 'survival time' is denoted as RMPB-II.

4.2.1 Baseline Fitness Functions

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

$$f_t^a(\mathbf{x}) = \frac{1}{d}\sum_{j=1}^{d} \max_{i=1}^{i=m}\{h_t^{ij} - w_t^{ij} * |x_j - c_t^{ij}|\}\,, \tag{4.6}$$

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

$$f_t^s(\mathbf{x}) = \min_{j=1}^{j=d}\{\max_{i=1}^{i=m}\{h_t^{ij} - w_t^{ij} * |x_j - c_t^{ij}|\}\}\,, \tag{4.7}$$

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

The construction of the two baseline fitness functions is motivated as follows. On the one hand, we would like both baseline fitness functions to be multi-

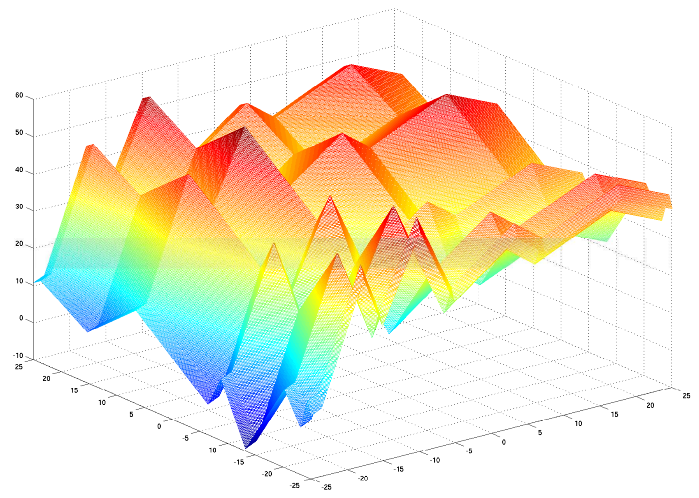


Figure 4.1: One fitness landscape example with 5 peaks along each dimension generated by the baseline fitness function in RMPB-I.

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

We generate two examples of fitness landscape using the baseline fitness function in Equation 4.6 in Figures 4.1 and 4.2. We generate another two examples of fitness landscape using the baseline fitness function in Equation 4.7 in Figures 4.3 and 4.4. In Figures 4.1 and 4.3, we have 5 peaks along each dimension (2 dimensional solution space), and we generate the landscapes randomly with each height ranging from 30 to 70, each width ranging from 1 to 13, and each centre ranging from -25 to 25. The same rule is applied to Figures 4.2 and 4.4 except that we set m, i.e., the number of peaks along each dimension, to 20. We can see that the fitness landscape in Figure 4.2 is more rugged than that in Figure 4.1, and so is the fitness landscape in Figure 4.4 than that in Figure 4.3.

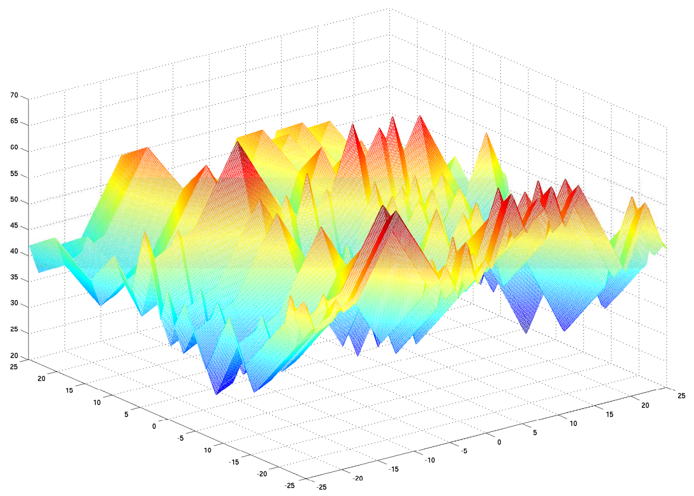


Figure 4.2: One fitness landscape example with 20 peaks along each dimension generated by the baseline fitness function in RMPB-I.

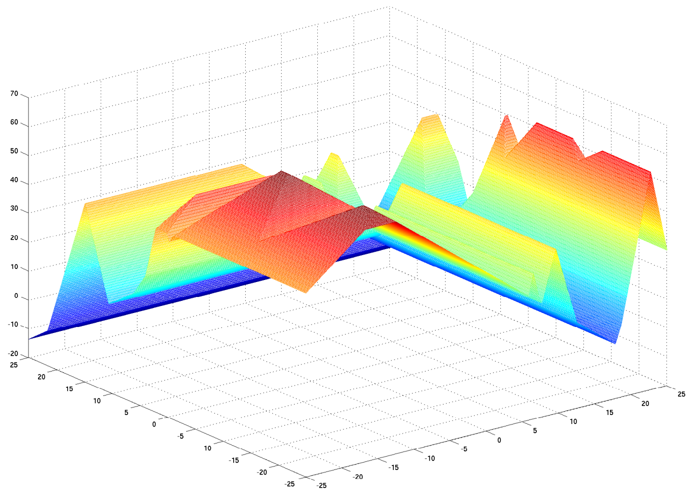


Figure 4.3: One fitness landscape example with 5 peaks along each dimension generated by the baseline fitness function in RMPB-II.

4.2.2 Dynamics for the Baseline Fitness Functions

We do not restrict the types of dynamics that are applied to the configure parameters, i.e., height h_t^{ij}, width w_t^{ij} and centre c_t^{ij}, in the two baseline fitness

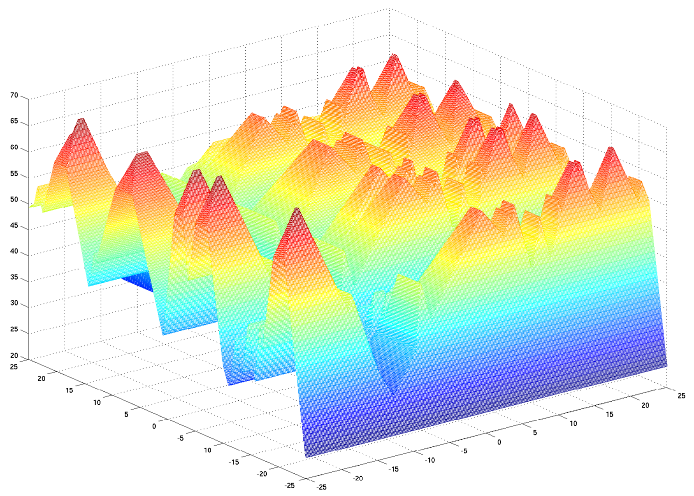


Figure 4.4: One fitness landscape example with 20 peaks along each dimension generated by the baseline fitness function in RMPB-II.

functions to generate DFFs. In other words, users can define their own dynamics for their study of ROOT problems using the two baseline fitness functions. However, we do suggest the following 6 types of dynamics that have been developed in CEC09 dynamic optimisation competition (Li et al. [2008]). The 6 different types of dynamics cover common dynamics found in real world DFFs, which are described as follows:

1. *small step*:
$$\Delta\phi = \gamma * ||\phi|| * r * \phi_{severity}, \tag{4.8}$$

2. *large step*:
$$\Delta\phi = ||\phi|| * (\gamma * sign(r) + (\gamma_{max} - \gamma) * r) * \phi_{severity}, \tag{4.9}$$

3. *random*:
$$\Delta\phi = \mathcal{N}(0, 1) * \phi_{severity}, \tag{4.10}$$

4. *chaotic*:

$$\phi_{t+1} = \phi_{min} + A * (\phi_t - \phi_{min}) * (1 - (\phi_t - \phi_{min})/||\phi||), \tag{4.11}$$

5. *recurrent*:

$$\phi_t = \phi_{min} + ||\phi|| * (sin(\frac{2\pi}{P}t + \varphi) + 1)/2, \tag{4.12}$$

6. *recurrent with noise*:

$$\phi_t = \phi_{min} + ||\phi|| * (sin(\frac{2\pi}{P}t + \varphi) + 1)/2 + \mathcal{N}(0,1) * noise_{severity}, \tag{4.13}$$

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

The height h_t^{ij} and width w_t^{ij} in Equations 4.6 and 4.7 are updated using the above mentioned 6 dynamics. An additional technique using rotation matrix is employed to rotate the centres (vector $\mathbf{c}_t^i = (c_t^{i1}, c_t^{i2}, ..., c_t^{id})$ in Equations 4.6 and 4.7). More specifically, according to (Li et al. [2008]), the following algorithm is used to change the centre:

1. l (l is even) number of dimensions are randomly selected from the d dimensions resulting a vector $(d_1, ..., d_l)$.

2. For each pair of dimension d_i and dimension d_{i+1} $(1 \le i \le l - 1)$, construct

a rotation matrix $\mathbf{R}_t(d_i, d_{i+1})$ that rotates the vector of centres in the plane $d_i - d_{i+1}$ by an angle θ_t from the d_i-th axis to the d_{i+1}-th axis.

3. Since rotation matrices are orthogonal, an overall rotation matrix $\mathbf{R}_t$ is obtained via: $\mathbf{R}_t = \mathbf{R}_t(d_1, d_2) * ... * \mathbf{R}_t(d_{l-1}, d_l)$.

4. The new vector of centres is produced by setting $\mathbf{c}_{t+1}^i = \mathbf{c}_t^i * \mathbf{R}_t$.

Moreover, the rotation angle θ_t is subject to the 6 different dynamics. Therefore, every time the DFF changes, the rotation angle θ_t is updated first, and the new angle is used to construct the rotation matrix that eventually changes the position of each centre.

Additionally, we introduce a parameter Δe for both RMPB-I and RMPB-II. Δe is measured in the number of fitness evaluations and is used to measure the computational budget for the decision maker to determine a solution right after an environmental change in ROOT problems. It should be noted that Δe should be generally smaller than the frequency of environmental changes measured in the number of fitness evaluations. The reason to introduce Δe, instead of a parameter that controls how frequently the DFF changes, is that usually a solution has to be found before a certain deadline in many real world DOPs (Δe is called the deadline parameter on Page 67 in Chapter 3 of (Nguyen [2011])). The deadline is usually before the next possible environmental change. In other words, solutions usually have to be determined before the next environmental change happens.

4.2.3 Relationship Between the Benchmark Parameters and ROOT Problem Difficulties

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

4.2.4 Calculating the Absolute Best 'Average Fitness' in RMPB-I

In this subsection, we prove two lemmas and one theorem for the purpose of calculating the absolute best 'average fitness' in Equation 4.1 for DFFs generated in RMPB-I.

Suppose a sequence of fitness functions $(f_1^a, f_2^a, f_3^a, ..., f_N^a)$ has been generated in RMPB-I. Without loss of generality, we would like to calculate the following absolute best 'average fitness': $\max\{F^a(\mathbf{x}, t, T)|\mathbf{x} \in \mathcal{S}\}$, which starts from f_t^a in the sequence $(f_1^a, f_2^a, f_3^a, ..., f_N^a)$ ($t < N$, $t + T - 1 \leq N$, and $T \geq 2$). $\max\{F^a(\mathbf{x}, 1, T)|\mathbf{x} \in \mathcal{S}\}$ takes the form as:

$$\max\{F^a(\mathbf{x}, t, T)|\mathbf{x} \in \mathcal{S}\} = \max\{\frac{1}{T}\sum_{i=0}^{T-1} f_{t+i}^a(\mathbf{x})|\mathbf{x} \in \mathcal{S}\}. \tag{4.14}$$

Firstly, we define the Maximal Average Fitness (MAF) of a number of fitness functions $(g_1, g_2, ..., g_k)$ as follows:

$$MAF(g_1, g_2, ..., g_k) = \max\{\frac{1}{k}\sum_{i=1}^{k} g_i(\mathbf{x})|\mathbf{x} \in \mathcal{S}\}, \tag{4.15}$$

where g_i, $1 \leq i \leq k$, represents a fitness function of $\mathbf{x}$. All the fitness functions, i.e., g_i, $1 \leq i \leq k$, share the same solution space $\mathcal{S}$. By definition, $\max\{F^a(\mathbf{x}, t, T)|\mathbf{x} \in \mathcal{S}\} = MAF(f_t^a, f_{t+1}^a, ..., f_{t+T-1}^a)$. We use $peak_t^{ij}$ to denote the ith peak function for the jth dimension at time step t in Equation 4.6: $peak_t^{ij} = h_t^{ij} - w_t^{ij} * |x_j - c_t^{ij}|$, $a \leq x_j \leq b$. We have:

Lemma 4.2.1. *The MAF for a set of peak functions* $\{peak_{t+k}^{i_k j}|0 \leq k \leq T-1\}$, *i.e.,* $MAF(peak_t^{i_0 j}, peak_{t+1}^{i_1 j}, ..., peak_{t+T-1}^{i_{T-1} j})$, *can be achieved when* x_j *takes the value of one of the centres* $c_{t+k}^{i_k j}$, $0 \leq k \leq T-1$.

Lemma 4.2.1 is proved in Appendix .1. From Lemma 4.2.1, we can see that the MAF for a set of peak functions $\{peak_{t+k}^{i_k j}|0 \leq k \leq T-1\}$ equals

$\max\{\frac{1}{T}\sum_{k=0}^{T-1}(h_{t+k}^{i_k j} - w_{t+k}^{i_k j} * |x_j - c_{t+k}^{i_k j}|) \mid x_j \in \{c_{t+k}^{i_k j}|0 \le k \le T-1\}\}$, which can be calculated by enumerating all the centres.

We use dim_t^j to denote the jth dimensional function at time step t in Equation 4.6: $dim_t^j = \max_{i=1}^{i=m}\{h_t^{ij} - w_t^{ij} * |x_j - c_t^{ij}|\}$, $a \le x_j \le b$. We have the following lemma, which is proved in Appendix .2:

Lemma 4.2.2. *The MAF of a set of dimensional functions* $\{dim_{t+k}^j|0 \le k \le T-1\}$*, i.e.,* $MAF(dim_t^j, dim_{t+1}^j, ..., dim_{t+T-1}^j)$*, is equal to* $\max\{MAF(peak_t^{i_0 j}, peak_{t+1}^{i_1 j}, ..., peak_{t+T-1}^{i_{T-1} j}) \mid 1 \le i_k \le m, 0 \le k \le T-1\}$.

Theorem 4.2.1. *The MAF of a set of fitness functions* $\{f_{t+k}^a|0 \le k \le T-1\}$ *is equal to* $\frac{1}{d}\sum_{j=1}^{d}\max\{MAF(peak_t^{i_0 j}, peak_{t+1}^{i_1 j}, ..., peak_{t+T-1}^{i_{T-1} j}) \mid 1 \le i_k \le m, 0 \le k \le T-1\}$.

Theorem 4.2.1 is proved based on Lemma 4.2.2 in Appendix .3. Based on Lemma 4.2.1 and Theorem 4.2.1, we can exactly calculate $MAF(f_t^a, f_{t+1}^a, ..., f_{t+T-1}^a)$, which is the absolute best 'average fitness', i.e., $\max\{F^a(\mathbf{x}, t, T)|\mathbf{x} \in \mathcal{S}\}$, given a sequence of fitness functions $(f_1^a, f_2^a, ..., f_N^a)$ generated in RMPB-I.

4.2.5 Calculating the Absolute Best 'Survival Time' in RMPB-II

In this subsection, we prove two lemmas and one theorem for the purpose of calculating the absolute best 'survival time' in Equation 4.2 for DFFs generated in RMPB-II.

Suppose a sequence of fitness functions $(f_1^s, f_2^s, f_3^s, ..., f_N^s)$ has been generated in RMPB-II. Without loss of generality, we would like to calculate the following absolute best 'survival time': $\max\{F^s(\mathbf{x}, t, V)|\mathbf{x} \in \mathcal{S}\}$, which starts from f_t^a $(1 \le t \le N)$ in the sequence $(f_1^a, f_2^a, f_3^a, ..., f_N^a)$ with the fitness threshold V.

Firstly, we define the Maximal Intersection Fitness (MIF) of a number of fitness functions $(g_1, g_2, ..., g_k)$ as follows:

$$MIF(g_1, g_2, ..., g_k) = \max\{\min_{i=1}^{i=k} g_i(\mathbf{x})|\mathbf{x} \in \mathcal{S}\}, \tag{4.16}$$

where g_i, $1 \le i \le k$, represents a fitness function of $\mathbf{x}$, and all the fitness functions, i.e., g_i, $1 \le i \le k$, share the same solution space $\mathcal{S}$. We use $peak_t^{ij}$ to denote the

ith peak function for the jth dimension at time step t in Equation 4.7: $peak_t^{ij} = h_t^{ij} - w_t^{ij} * |x_j - c_t^{ij}|$, $a \leq x_j \leq b$. It is easy to verify that the MIF of $peak_t^{i_0 j}$ and $peak_{t+1}^{i_1 j}$, i.e., $MIF(peak_t^{i_0 j}, peak_{t+1}^{i_1 j})$, is equal to:

$$\begin{cases} h_t^{i_0 j}, & \text{if } h_{t+1}^{i_1 j} - w_{t+1}^{i_1 j} * |c_t^{i_0 j} - c_{t+1}^{i_1 j}| \geq h_t^{i_0 j}, \\ h_{t+1}^{i_1 j}, & \text{elseif } h_t^{i_0 j} - w_t^{i_0 j} * |c_{t+1}^{i_1 j} - c_t^{i_0 j}| \geq h_{t+1}^{i_1 j}, \\ \frac{w_t^{i_0 j} * h_{t+1}^{i_1 j} + w_{t+1}^{i_1 j} * h_t^{i_0 j} - w_t^{i_0 j} * w_{t+1}^{i_1 j} * |c_t^{i_0 j} - c_{t+1}^{i_1 j}|}{w_t^{i_0 j} + w_{t+1}^{i_1 j}}, & \text{else.} \end{cases} \tag{4.17}$$

For a number of peak functions, we have the following lemma, which is proved in Appendix .4:

Lemma 4.2.3. *The MIF of a set of peak functions* $\{peak_{t+k}^{i_k j} | 0 \leq k \leq L-1\}$*, i.e.,* $MIF(peak_t^{i_0 j}, peak_{t+1}^{i_1 j}, ..., peak_{t+L-1}^{i_{L-1} j})$*, is equal to* $\min\{MIF(peak_{t+p}^{i_p j}, peak_{t+q}^{i_q j}) | 0 \leq p < q \leq L-1\}$.

We use dim_t^j to denote the jth dimensional function at time step t in Equation 4.7: $dim_t^j = \max_{i=1}^{i=m}\{h_t^{ij} - w_t^{ij} * |x_j - c_t^{ij}|\}$, $a \leq x_j \leq b$. We have the following lemma, which is proved in Appendix .5:

Lemma 4.2.4. *The MIF of a set of dimensional functions* $\{dim_{t+k}^j | 0 \leq k \leq L-1\}$*, i.e.,* $MIF(dim_t^j, dim_{t+1}^j, ..., dim_{t+L-1}^j)$*, is equal to* $\max\{MIF(peak_t^{i_0 j}, peak_{t+1}^{i_1 j}, ..., peak_{t+L-1}^{i_{L-1} j}) \mid 1 \leq i_k \leq m, 0 \leq k \leq L-1\}$.

Theorem 4.2.2. *The MIF of a set of fitness functions* $\{f_{t+k}^s | 0 \leq k \leq L-1\}$*, i.e.,* $MIF(f_t^s, f_{t+1}^s, ..., f_{t+L-1}^s)$*, is equal to* $\min_{j=1}^d\{\max\{MIF(peak_t^{i_0 j}, peak_{t+1}^{i_1 j}, ..., peak_{t+L-1}^{i_{L-1} j}) \mid 1 \leq i_k \leq m, 0 \leq k \leq L-1\}\}$.

Theorem 4.2.2 is proved based on Lemma 4.2.4 in Appendix .6. Based on Equation 4.17, Lemma 4.2.3, and Theorem 4.2.2, we can exactly calculate $MIF(f_t^s, f_{t+1}^s, ..., f_{t+L-1}^s)$ for any L number of consecutive fitness functions in a sequence of fitness functions $(f_1^a, f_2^a, ..., f_N^a)$ generated in RMPB-II. The absolute best 'survival time', $\max\{F^s(\mathbf{x}, t, V) | \mathbf{x} \in \mathcal{S}\}$, starting from f_t^s, is equal to the largest number that L can take satisfying $MIF(f_t^s, f_{t+1}^s, ..., f_{t+L-1}^s) \geq V$.

4.3 Behaviour of Existing Methods for ROOT

In this section, experimental studies are conducted regarding the performance of existing methods on RMPB-I and RMPB-II. The purpose is to investigate the strengths and weaknesses of existing methods on ROOT problems. Also, we would like to examine the performance of existing methods on ROOT by showing the gaps between the performance of different methods and the absolute best performance in our benchmarks.

4.3.1 Experimental Settings

4.3.1.1 Test Problems

We generate 6 benchmark instances in RMPB-I by applying the 6 types of dynamics defined in Equations 4.8 to 4.13 to the first baseline fitness function in Equation 4.6. In the meantime, we apply those 6 types of dynamics to the second baseline fitness function in Equation 4.7 to produce 6 benchmark instances in RMPB-II. We refer each of them as TP_{ij}, $i = 1, 2$, $j = 1, 2, ..., 6$, where TP_{ij} means the benchmark instance that is generated by applying the jth dynamic to the ith baseline fitness function. TP_{1j}, $j = 1, 2, ..., 6$, are used for maximizing 'average fitness' in ROOT, and TP_{2j}, $j = 1, 2, ..., 6$, are used for maximizing 'survival time' in ROOT. For each TP_{ij}, we generate 200 consecutive fitness functions as follows. For the first fitness function, we randomly initialize the heights and the widths in their corresponding ranges, and the centres are randomly initialized across the solution space. To generate the next fitness function, we apply the jth dynamic to the current heights, widths, and rotation angle. The centres of the next fitness function are obtained by rotating the centres of the current fitness function using the updated rotation angle. Whenever the heights, widths, or centres get out of their corresponding ranges, we reset them to their up limits (if larger than up limits) or low limits (if lower than low limits). Note the following exceptions. For the *chaotic* change, centres are updated dimension by dimension using Equation 4.11 rather than the rotation technique. For the *recurrent* change and the *recurrent with noise* change, θ is fixed to $\frac{2\pi}{P}$ where P is the period, and the centres are rotated following a fixed direction. A summary of all test problem parameters is presented in Table 4.1.

Table 4.1: Parameter settings of the test problems.

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

It should be emphasized that both T and V are ROOT problem parameters. T is used to indicate how many time steps a robust solution in terms of 'average fitness' is kept in use. V indicates the minimal fitness that a solution in terms of 'survival time' is required to maintain in order to be robust in a particular situation. In this thesis, we simply set T and V to some fixed values that have a certain range.

4.3.1.2 Methods Investigated for ROOT Problems

We select 4 representative methods from the literature to test their ROOT abilities. The first approach is a simple PSO with a re-start strategy, which we will denote as 'RPSO' hereafter. The re-start strategy means that the best solution found in terms of the fitness for the last fitness function at the last time step is copied into the initial population for the current time step whenever the environment changes and that all other particles are initialized randomly. The second approach can be seen as an ideal TMO algorithm, in which each fitness function's optimum is chosen as the ROOT solution at each time step. The ideal TMO approach, denoted as 'optimum' hereafter, can be viewed as the best any TMO approach can do in terms of TMO. The third and fourth approaches are two latest methods developed specially for ROOT problems. The third approach is Jin et al.'s framework (Jin et al. [2012]), denoted as 'Jin's' hereafter. A global Radial Basis Function Network (RBFN) is employed as the surrogate model to approximate a solution's previous fitness in 'Jin's'. For predicting a solution's future fitness in 'Jin's', the AR (Box et al. [1994]) model with order 4 is employed. One estimated previous fitness and four predicted future fitnesses are used in the metric[1] in 'Jin's', the setting of which is reported to have the best performance in (Jin et al. [2012]). For more details of 'Jin's', readers are referred to (Jin et al. [2012]). The fourth approach is the method developed by us in (Fu et al. [2013]), which we will denote as 'Fu's' hereafter. The same RBFN and AR models are used in 'Fu's' as in 'Jin's' except that the metrics used for respectively maximizing 'average fitness' and 'survival time' in 'Fu's' are different from the metric in 'Jin's'. Furthermore, the weight coefficient, which is associated with the accuracy of the estimator in the metrics in 'Fu's', is set to 1. For more details of 'Fu's', readers are referred to (Fu et al. [2013]).

For methods 'RPSO', 'Jin's', and 'Fu's', they all employ the same constriction version of PSO (Clerc and Kennedy [2002]) as the optimizer. The swarm population size is 50. The constants c_1 and c_2 that are used to bias a particle's attraction to the personal best and the global best are both set to 2.05, and therefore the constriction factor χ takes the value 0.729844. The velocity of particles is constricted within the range $[-v_{max}, v_{max}]$. The value of v_{max} is set to the range of the search space, which is 50 in our case.

We believe it is necessary to test TMO methods for the purpose of ROOT. For the current time step, the objective of TMO is to find a solution maximizing the current fitness function. In contrast, the objective of ROOT is to find a solution which maximizes 'average fitness' or 'survival time'. It is easy to note

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

that a solution's current fitness does not necessarily contradict with the solution's 'average fitness' or 'survival time'. In other words, whether a TMO method would be successful in finding robust solutions in ROOT largely depends on how the DFF changes over time, and the question is under what circumstances will TMO methods be effective or not in solving ROOT problems. The reason to employ 'RPSO' and 'Optimum' methods is that they serve to represent a vast majority of approaches designed for the purpose of TMO. The ideal TMO approach, i.e., 'optimum' approach, is selected in the hope that the effort to enumerate all state-of-the-art TMO approaches can be saved. Actually, 'optimum' approach is the best any algorithm can do in terms of TMO. Besides, the reason to include 'Jin's' is that it is, to the best of our knowledge, the only method from the literature that has been designed specially for ROOT problems so far.

4.3.1.3 Performance Measurement

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

$$Performance^{ROOT} = \frac{1}{N}\sum_{i=1}^{N} F(\mathbf{x}_i), \tag{4.18}$$

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

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

4.3.2 Experimental Results

All the results presented below are for time steps 20 to 100, i.e., from the 20th fitness function to the 100th fitness function in the sequence of TP_{ij}, $i = 1$, 2 , $j = 1, 2, ..., 6$. The reason is that we require 20 length of time series data available for the prediction of a solution's future fitness in 'Jin's' and 'Fu's'. This means the performance measure in Equation 4.18 is calculated by averaging the corresponding robustness of the solution determined by the investigated method at each time step (from time step 20 to 100). Since some solution's 'survival time' can be infinity in certain benchmarks, we reset a solution's 'survival time' to be 10 if its 'survival time' is larger than 10.

Results with a '+' or '-' attached in the right hand side in Tables 4.2 and 4.3 are significantly better or worse than those of 'Fu's' at a 0.05 significance level of Wilcoxon rank sum test.

4.3.2.1 Results for Maximizing 'Average Fitness'

The results of maximizing 'average fitness' in ROOT using the four methods are presented in Table 4.2. We can see that when the time window T takes a small value, i.e., $T = 2$, it is generally better to use TMO methods rather than methods specially designed for ROOT problems (i.e., 'Jin's' and 'Fu's'). The reason is straightforward as within a small time window T, previously good solutions tend to remain good during that time window. However, as the time window T gets larger ($T = 6, 10$), the advantage of ROOT methods (i.e., 'Jin's' and 'Fu's', especially 'Fu's') over TMO methods gradually shows up. 'Fu's' is significantly better than 'RPSO' and 'Optimum' in almost all cases when $T = 6, 10$. However, for some specific dynamics in the DFFs, such as the *random* change in TP_{13} and the *recurrent with noise* change in TP_{16}, 'RPSO' and 'Optimum' outperform 'Jin's' and 'Fu's'. The reason is that there is randomness in the change of the

DFF in TP_{13} or TP_{16}, and hence prediction methods in 'Jin's' and 'Fu's' are inaccurate, which degrades the performance of 'Jin's' and 'Fu's'. Finally, for the comparison between 'Jin's' and 'Fu's', 'Fu's' outperforms 'Jin's' in most cases. The success of 'Fu's' over 'Jin's' is primarily due to the metrics in 'Fu's', which guide EAs better to converge to good solutions. For more details of 'Fu's', readers are referred to (Fu et al. [2013]).

Table 4.2: The average performance in Equation 4.18 over 30 runs under the robustness definition of 'average fitness' with different settings of time window T.

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

4.3.2.2 Results for Maximizing 'Survival Time'

The results for maximizing 'survival time' in ROOT for methods 'RPSO', 'Optimum', 'Jin's', and 'Fu's' are presented in Table 4.3. We can see that 'Fu's' outperforms 'RPSO', 'Optimum', and 'Jin's' in most cases except in some cases when the fitness threshold V takes a large value, say $V = 50$. The reason why 'Optimum' outperforms 'Fu's' in some cases when $V = 50$ is that there exist few solutions which can maintain their fitness above V for more than 2 time steps if the fitness threshold V is set high. In other words, maximizing 'survival time' in ROOT is reduced to TMO when the fitness threshold V is set relatively high. On the other hand, even in some randomly changing DFFs, such as in TP_{23}, 'Fu's' still outperforms 'RPSO' and 'Optimum' when $V = 40, 45$. The reason is that, even though the prediction in 'Fu's' is inherently inaccurate, the prediction method in 'Fu's' can still estimate the uncertainty in a solution's predicted future fitness, which is helpful in estimating a solution's 'survival time'. In contrast, TMO methods only maximize a solution's current fitness, without any consideration of a solution's future fitness. Finally, 'Fu's' outperforms 'Jin's' in most cases, and the primary reason is again due to the metrics used in 'Fu's'. For more details of 'Fu's', readers are referred to (Fu et al. [2013]).

Table 4.3: The average performance in Equation 4.18 over 30 runs under the robustness definition of 'survival time' with different settings of fitness threshold V.

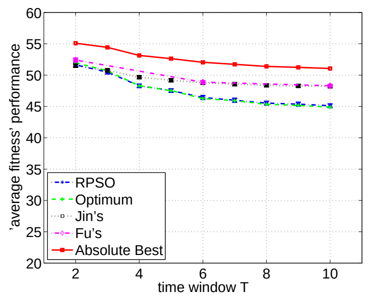
Test	RPSO			Optimum			Jin's			Fu's		
Problem	$V=40$	$V=45$	$V=50$	$V=40$	$V=45$	$V=50$	$V=40$	$V=45$	$V=50$	$V=40$	$V=45$	$V=50$
TP_{21}	2.09-	1.60-	1.12-	1.94-	1.48-	1.11-	3.19-	1.84-	0.91-	3.55	2.31	1.26
TP_{22}	1.10-	1.02	0.96+	1.11-	1.02	0.98+	0.78-	0.56-	0.45-	1.13	1.03	0.95
TP_{23}	1.29-	1.12-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

4.3.2.3 Gaps Between the Absolute Best and the Results obtained by Investigated Methods

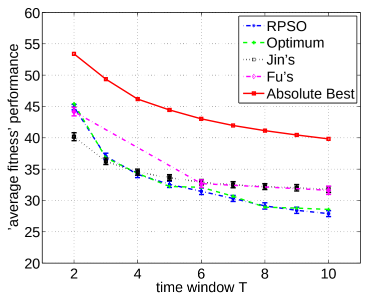
In Tables 4.2 and 4.3, we have shown the ROOT performance of investigated methods on TP_{ij}, $i = 1, 2, 1 \leq j \leq 6$. We would also like to know the gaps between the absolute best performance and the performance of any investigated method for solving ROOT problems.

We plot the absolute best performance in terms of 'average fitness', calculated using Theorem 4.2.1, and the corresponding performance of investigated methods on TP_{1j} under different settings of time window T, $T = 2, 3, 4, ..., 10$, in Figure 4.5. It should be noted that 'Fu's' has only been tested on $T = 2, 6, 10$. Taking the subfigure Figure 4.5(a) for example, we calculate the absolute best performance in the following way (suppose $T = 2$). For TP_{11}, we have already generated a sequence of fitness functions of length 200, say $(f_1^a, f_2^a, ..., f_{200}^a)$. We then calculate the absolute best 'average fitness' in Equation 4.1 for each fitness function in the sequence $(f_{20}^a, f_{21}^a, ..., f_{100}^a)$ based on Theorem 4.2.1. Finally, we average the absolute best 'average fitness' for each fitness function in the sequence $(f_{20}^a, f_{21}^a, ..., f_{100}^a)$ to obtain the absolute best performance for TP_{11} with $T = 2$. Other absolute best performance for TP_{1j} $(1 \leq j \leq 6)$, with $T = 2, 3, ..., 10$, are produced in the same way. From Figure 4.5, we can see that the gaps between the absolute best performance and the performance of investigated methods get larger as T increases. In other words, the difficulty in maximizing 'average fitness' in ROOT increases as T gets larger.

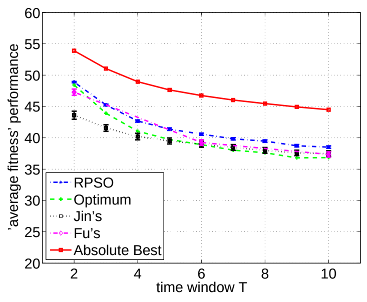
We plot the absolute best performance in terms of 'survival time', calculated using Theorem 4.2.2, and the corresponding performance of investigated methods



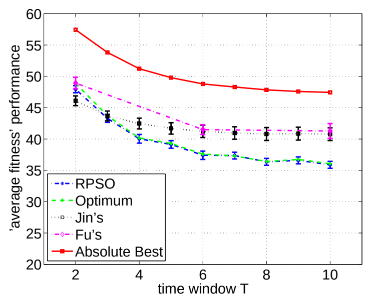
(a) TP_{11}



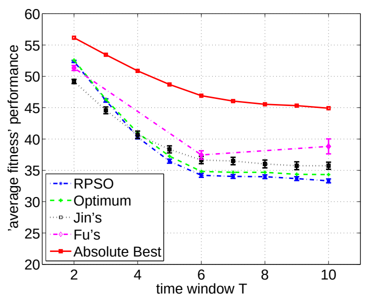
(b) TP_{12}



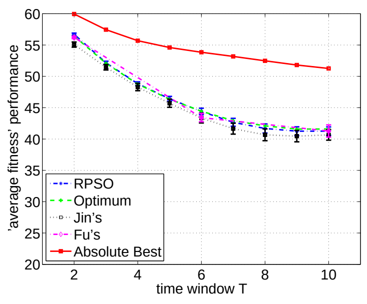
(c) TP_{13}



(d) TP_{14}



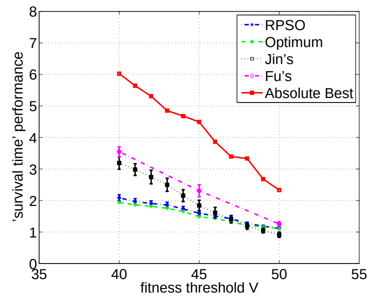
(e) TP_{15}



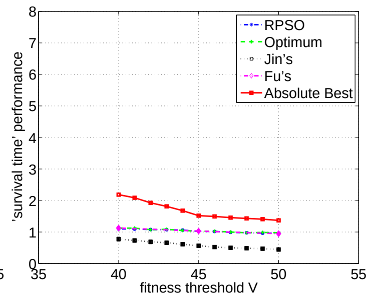
(f) TP_{16}

Figure 4.5: The average performance in Equation 4.18 with one standard deviation errorbar over 30 runs of investigated methods under the robustness definition of 'average fitness' with different settings of time window T, in comparison with the corresponding absolute best performance.

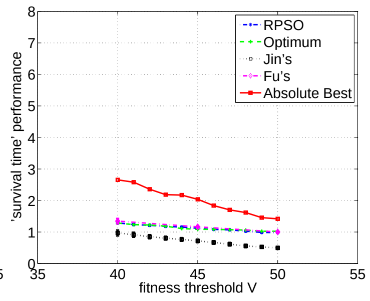
on TP_{2j} under different settings of fitness threshold V, $V = 40, 41, 42, ..., 50$, in Figure 4.6. It should be noted that 'Fu's' has only been tested on $V = 40, 45, 50$. Taking the subfigure Figure 4.6(a) for example, we calculate the absolute best performance in the following way (suppose $V = 40$). For TP_{21}, we have already generated a sequence of fitness functions of length 200, say $(f_1^s, f_2^s, ..., f_{200}^s)$. We then calculate the absolute best 'survival time' in Equation 4.2 for each fitness function in the sequence $(f_{20}^s, f_{21}^s, ..., f_{100}^s)$ based on Theorem 4.2.2. Finally, we average the absolute best 'survival time' for each fitness function in the sequence $(f_{20}^s, f_{21}^s, ..., f_{100}^s)$ to obtain the absolute best performance for TP_{21} with $V = 40$. Other absolute best performance for TP_{2j} $(1 \leq j \leq 6)$, with $V = 40, 41, 42, ..., 50$, are produced in the same way. From Figure 4.6, we can see that the gaps between the absolute best performance and the performance of investigated methods get larger as V decreases. In other words, the difficulty in maximizing 'survival time' in ROOT increases as V gets smaller.



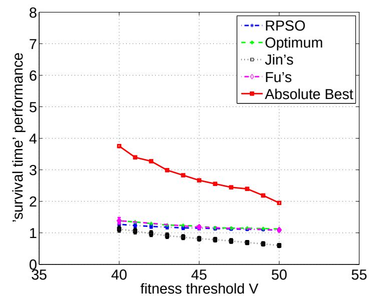
(a) TP_{21}



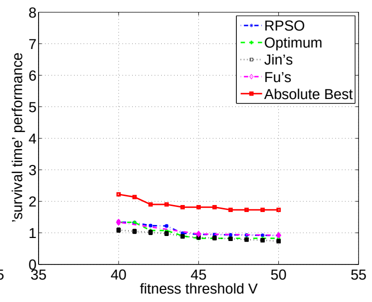
(b) TP_{22}



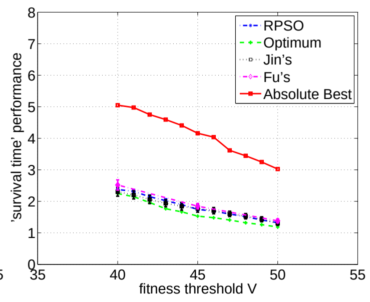
(c) TP_{23}



(d) TP_{24}



(e) TP_{25}



(f) TP_{26}

Figure 4.6: The average performance in Equation 4.18 with one standard deviation errorbar over 30 runs of investigated methods under the robustness definition of 'survival time' with different settings of fitness threshold V, in comparison with the corresponding absolute best performance.

4.4 Summary

In this chapter, we have investigated the benchmark design of ROOT. Existing DOP benchmarks were briefly reviewed, and their problems for using as ROOT benchmarks were discussed. We argued that it is difficult to calculate the absolute best 'average fitness' and the absolute best 'survival time' on existing DOP benchmarks. Therefore, it is not sufficient to test an algorithm's ROOT ability on existing DOP benchmarks as the lack of knowledge about the absolute best performance makes the evaluation and comparison of algorithms on existing benchmarks difficult if not impossible.

The primary contribution of this chapter is the two benchmark problems, i.e., RMPB-I and RMPB-II, specially designed for maximizing 'average fitness' and 'survival time' in ROOT respectively. Particularly, lemmas and theorems were developed with regard to the calculation of the absolute best 'average fitness' in RMPB-I and the absolute best 'survival time' in RMPB-II.

The second contribution of this chapter is the experimental studies presented in this chapter. Several methods from the literature were tested on the two ROOT benchmarks, which demonstrates the potential strengths and weaknesses of different methods for ROOT problems with different dynamics. Besides, it was shown that the ROOT problem parameters (i.e., the time window T when maximizing 'average fitness' or the fitness threshold V when maximizing 'survival time') also have a huge impact on an algorithm's performance. To be more specific, the difficulty in maximizing 'average fitness' increases as T increases, and the difficulty in maximizing 'survival time' increases as V decreases. More importantly, the difficulties of ROOT problems in terms of maximizing 'average fitness' or 'survival time' were demonstrated for the first time by showing the gaps between the absolute best performance and the performance of investigated methods.

The two ROOT benchmarks developed in this chapter do not have time-linkage. For the future work, it would be interesting to incorporate some time-linkage into the benchmark problems as time-linkage is found in many real world DOP applications (Bosman [2005], Bosman and La Poutre [2007], Nguyen [2011], Nguyen and Yao [2013]). Moreover, the constraints in the solution space of the two ROOT benchmarks stay constant over time, which may not be the case in some real world DOP applications as discussed in the dynamic constraint optimisation (Nguyen [2011]). Therefore, it would be interesting to extend the two ROOT benchmarks by varying the constraints over time.

CHAPTER 5

LEARNING AND OPTIMIZING IN ROBUST OPTIMISATION OVER TIME

In Chapter 4, we developed two types of benchmark problems for respectively maximizing ‘average fitness’ and ‘survival time’ in ROOT, in which the absolute best performance is available. Moreover, existing methods for ROOT were tested on our benchmarks, which demonstrated the strengths and weaknesses of existing methods for ROOT problems under different dynamics. In this chapter, we try to develop an algorithm framework for ROOT that handles different dynamics properly. The rest of this chapter is organised as follows. Existing methods for ROOT are briefly reviewed in Section 5.1, with some discussions on their potential problems for ROOT. Based on the discussions, a novel algorithm framework is then developed in Section 5.2. Some experiments are conducted in Section 5.3, evaluating the effectiveness of our algorithm framework against existing methods on maximizing ‘average fitness’ and ‘survival time’ in ROOT. Finally, a summary is given in Section 5.4.

5.1 Existing Methods for ROOT

Since the idea of ROOT was proposed just a few years ago (Yu et al. [2010]), only a few methods have been developed specially for ROOT. The first method developed specially for ROOT is the algorithm framework in (Jin et al. [2012]). The framework consists of an optimizer for searching for ROOT solutions, a component for estimating a solution’s previous fitness, and a component for predicting a solution’s future fitness. Within the optimizer, a solution is evaluated by the metric that is the average of its past, current, and future fitness. To be more specific, a solution's previous fitness is approximated by building a surrogate model using neighbourhood solutions evaluated at the same previous time step, while the solution's current fitness is evaluated using the current fitness function. By modelling a solution's past and current fitness as a time series, the solution's future fitness is predicted by training a learning model on the time series. One problem with the method in (Jin et al. [2012]) for ROOT is that the metric in the optimizer does not have a direct connection with a solution's robustness definition. Later on, we extended the framework in (Jin et al. [2012]) by developing two metrics that are used for maximizing two corresponding robustness definitions (i.e., 'average fitness' and 'survival time' in this thesis) in (Fu et al. [2013]). One problem with the method in (Fu et al. [2013]) and also with the method in (Jin et al. [2012]) is the computational cost incurred for estimating a solution's previous fitness. To be more specific, whenever a solution is evaluated in (Jin et al. [2012]) or (Fu et al. [2013]), $L-1$ number of surrogate models have to be built, where L is the solution's fitness time series length. Building a surrogate model itself is usually a computationally costly task (Jin [2011]). As a result, the overall computational cost for estimating a solution's previous fitness can be unbearable if a large number of fitness evaluations are conducted at a time step. Another problem with the method in (Fu et al. [2013]) and also with the method in (Jin et al. [2012]) lies in the part for predicting a solution's future fitness. A single learning model is used for prediction, whose performance can be unstable and unsatisfactory as we often do not know the dynamic of the DFF we deal with.

Aside from the methods developed specially for ROOT, various methods have been developed for solving different types of DOPs, and they may be potentially used for the purpose of ROOT. Based on the review of different types of DOPs in EDO in Chapter 2, we can see that methods that are developed for the purpose of TMO can also be used for solving ROOT problems. The reason is that both TMO and ROOT aim to find an optimal solution at every time step, except that the optimality in TMO is different from that in ROOT. At each time step, the optimality in TMO means maximizing the current fitness function, while the optimality in ROOT means maximizing a corresponding robustness definition. It is possible for a solution being optimal both in TMO and ROOT, depending on how the DFF changes. However, it is easy to understand that TMO methods may be suitable for ROOT problems only when a good solution in terms of the current fitness is also good in terms of the corresponding robustness.

5.2 A Novel Algorithm Framework for ROOT

In this section, we develop a new approach for solving ROOT problems, which is a generic algorithm framework for finding ROOT solutions repeatedly over time. Within the framework, a population-based search algorithm is employed to search for ROOT solutions based on the metrics specially designed for the corresponding robustness definition of solutions to ROOT problems. Two metrics are proposed to respectively maximize 'average fitness' and 'survival time'. The metrics are functions of a solution's current and future fitness. It should be noted that a solution's robustness is evaluated based on its true future fitness, while its metric value is calculated based on its predicted future fitness. Hence, learning models are built to predict a solution's future fitness within the framework. More specifically, we build surrogate models to estimate a solution's fitness at previous time steps, based on which we construct a time series of the solution's fitness over time. Ensemble learning is then used to predict the solution's future fitness based on the fitness time series. In contrast to existing approaches for ROOT problems, the main contributions of our approach are the two new metrics, the mechanism for building the surrogate models, and the method for predicting a solution's future fitness via ensemble learning. Our algorithm framework is described in detail in the following.

5.2.1 Ensemble Learning for ROOT

One of the central tasks in ROOT is to predict a solution's future fitness (Jin et al. [2012], Fu et al. [2013]). In the ideal case where a solution's future fitness can be perfectly predicted, a ROOT problem will degenerate into a TMO problem as we can exactly evaluate a solution's robustness (a solution's robustness is a function of its current and future fitness) like we evaluate its current fitness. However, prediction in ROOT is a nontrivial task. In our algorithm framework, we model a solution's fitness over time as a time series. Suppose at time step t, we have got a fitness time series regarding to solution $\mathbf{x}$: $(f_{t-L+1}(\mathbf{x}), ..., f_t(\mathbf{x}))$ where $f_i(\mathbf{x})$, $t - L + 1 \leq i \leq t$, is the fitness of solution $\mathbf{x}$ when the DFF took the form of f_i, and L is the length of the time series. We propose to use multiple learning models, i.e., an ensemble, to predict $\mathbf{x}$'s future fitness based on $\mathbf{x}$'s fitness time series data as follows:

$$\hat{f}_{t+j}(\mathbf{x}) = \sum_{i=1}^{k} [\beta_t^i(\mathbf{x}) * \hat{f}_{t+j}^i(\mathbf{x})], \tag{5.1}$$

where $\hat{f}_{t+j}^i(\mathbf{x})$ is the prediction of the ith learner for solution $\mathbf{x}$'s future fitness at time step $t+j$, $j = 1, 2, ...$, and $\beta_t^i(\mathbf{x})$ is the weighting function for the ith learner at time step t. k is the number of learners in the ensemble. Accordingly, $\hat{f}_{t+j}(\mathbf{x})$ is the prediction of the ensemble for solution $\mathbf{x}$'s future fitness at time step $t + j$.

The reasons for using multiple learning models rather than a single model are as follows:

- It has been shown in the machine learning community that ensemble learning is able to improve the prediction accuracy in terms of Mean Square Error (MSE) compared to single model learning, given a diverse set of learners in the ensemble (Perrone and Cooper [1992]).

- The prediction accuracy of a solution's future fitness has a huge impact on the performance of population-based search algorithms in finding ROOT solutions. The reason is that if the prediction accuracy is so low that a population-based search algorithm thinks an actually better ROOT solution worse than the other, the population-based search algorithm would probably converge to a false optimum.

There has been work in the literature showing that ensembles can help to improve prediction performance over single learners. However, the extent to which ensembles can help varies from one problem to another, and the advantages of using ensembles may not be significant in some cases. Therefore, we would like to investigate for the first time to what extent ensembles could help over single learners in situations of ROOT problems, given that the training data in ROOT can be quite noisy (a solution's fitness time series consists of its estimated fitness at previous time steps).

We adopt a simple ensemble method for predicting a solution's future fitness in ROOT for its efficient use of training data and small time complexity. The ensemble method is called the basic ensemble method, which was proposed in (Perrone and Cooper [1992]). In the basic ensemble method, the output of ensemble is the average of outputs of each learner, which takes the form as follows:

$$\hat{f}_{t+j}(\mathbf{x}) = \sum_{i=1}^{k}[\frac{1}{k} * \hat{f}^{i}_{t+j}(\mathbf{x})], \tag{5.2}$$

where the same notations are used as in Equation 5.1. The basic ensemble method sets the weight $\beta^i_t(\mathbf{x})$ for each learner as a constant $\frac{1}{k}$. Each learner in the basic ensemble method is trained separately on the whole training data. Any prediction made by the basic ensemble method is the average of predictions made by each learner in the ensemble. It has been shown in (Perrone and Cooper [1992]) that the basic ensemble method reduces the MSE by a factor of k compared to the average MSE over each learner under mild assumptions. As for the selection of learners in the basic ensemble, we would like to select a diverse set of learners in the hope that the mutually independent error assumption will hold (Tang et al. [2006]).

5.2.2 Optimizing for ROOT

The objective in ROOT at time step t is not to find a solution maximizing the fitness function f_t, which is the theme of TMO approaches, but the corresponding robustness definition (i.e., 'average fitness' or 'survival time'). Both 'average fitness' and 'survival time' involve a solution's future fitness but differ in the way how a solution's current and future fitness contribute to the corresponding robustness. In the following, we describe the metrics, which are used in population-based search algorithms to search for optimal solutions in terms of the defined robustness, in the context of 'average fitness' and 'survival time' respectively.

5.2.2.1 Maximizing ‘Average Fitness’ in ROOT

In this case, a solution’s ‘average fitness’ over a consecutive number of time steps is considered. The consecutive number of time steps is specified by the user in the parameter T. Given a solution’s current fitness $f_t(\mathbf{x})$ and predicted future fitness $\hat{f}_{t+i}(\mathbf{x})$, $i = 1, 2, ...T - 1$, produced by our ensemble learning method, the metric for maximizing a solution’s ‘average fitness’ is:

$$\hat{F}^a(\mathbf{x}, t, T) = \frac{1}{T}(f_t(\mathbf{x}) + \sum_{i=1}^{T-1} \hat{f}_{t+i}(\mathbf{x})). \tag{5.3}$$

This metric was first proposed by us in (Fu et al. [2013]). If the predicted fitness $\hat{f}_{t+i}(\mathbf{x})$ can be interpreted as the expected fitness of solution $\mathbf{x}$ at time step $t+i$, the metric in Equation 5.3 is then the expected ‘average fitness’ of solution $\mathbf{x}$ over a time period specified by T.

5.2.2.2 Maximizing ‘Survival Time’ in ROOT

The metric for ‘survival time’ is not as straightforward as that for ‘average fitness’. In (Fu et al. [2013]), we employed a simple heuristic to develop a metric for maximizing ‘survival time’. Basically, for a solution $\mathbf{x}$, we first predict a number of its future fitness. After that, we simply count the number of consecutive time steps, during which $\mathbf{x}$’s fitness is above the fitness threshold V, as the metric. The problem with the metric for maximizing ‘survival time’ in (Fu et al. [2013]) is that it treats a higher predicted fitness and a lower predicted fitness the same as long as they are both above the fitness threshold V. However, from a probabilistic point of view, the higher predicted fitness is more likely to be above the threshold than the lower one if they have the same predicted variance. Based on this consideration, we employ a more probabilistic point of view to develop the metric for maximizing ‘survival time’ as follows:

$$\hat{F}^s(\mathbf{x}, t, V) = \begin{cases} f_t(\mathbf{x}) & \text{if } f_t(\mathbf{x}) < V, \\ V + \omega * \hat{l} & \text{otherwise,} \end{cases} \tag{5.4}$$

where a constant scaling factor ω is introduced to make V and $\hat{l}$ in the same scale[1]. $\hat{l}$ is the expected 'survival time' calculated as:

$$\sum_{i=1}^{L^p-1}[(i+1)*\prod_{j=1}^{i}(1-\Phi(\frac{V-\hat{f}_{t+j}(\mathbf{x})}{\sigma}))*\Phi(\frac{V-\hat{f}_{t+i+1}(\mathbf{x})}{\sigma})], \tag{5.5}$$

where $\Phi(y)$ returns the probability that the value of a standard normal random variable is smaller than y. We are assuming the predicted fitness of $\mathbf{x}$ at time step $t+i$, $1 \leq i \leq L^p$, follows a normal distribution with mean $\hat{f}_{t+i}$ and standard deviation σ. σ is calculated in our approach according to Equation 11 in Appendix .8. As a result, $\Phi(\frac{V-\hat{f}_{t+j}(\mathbf{x})}{\sigma})$ calculates the probability that $f_{t+j}(\mathbf{x})$ is smaller than V. L^p is a parameter that specifies the number of predicted future fitness used for the calculation of $\hat{l}$. In other words, the probability of a solution's 'survival time' greater than L^p is considered as null when calculating $\hat{l}$.

The metric in Equation 5.4 is motivated as follows. According to the definition of 'survival time' in Equation 3.3 in Section 3.3, a solution's 'survival time' from time step t is 0 if its fitness f_t is smaller than V. Supposing the current time step is t, the first goal therefore for the population-based search algorithm is to find solutions with f_t not smaller than V. Once a solution's f_t is not smaller than V, we then try to maximize the number of consecutive time steps during which this solution can maintain its fitness no smaller than V. The metric in Equation 5.4 is designed in such a way that a solution with fitness f_t no smaller than V is always considered better than any solution whose f_t is smaller than V. In contrast, solutions are compared based on the estimated 'survival time' when both have fitness f_t no smaller than V. Note that Equation 5.4 is a metric for maximizing 'survival time' in Equation 3.3 in Section 3.3. Therefore, Equation 5.4 does not necessarily have the same form as the definition of 'survival time' in Equation 3.3 in Section 3.3.

[1]The part $\hat{l}$ will be treated as 0 in computers if V is much larger than $\hat{l}$, and vice versa. The possible value of V is within the range $[-1000, 1000]$ since a solution's fitness in the benchmarks is within the range $[-1000, 1000]$, and thus ω is properly set to 1.

5.2.3 An Optimizing and Ensemble Learning Framework for ROOT

At each time step in ROOT, our goal is to find a solution that maximizes 'average fitness' or 'survival time'. Given a certain amount of computational resources at each time step, we are interested in the final solution the algorithm outputs. Without loss of generality, given the information of previously evaluated solutions in previous time steps, we describe our algorithm framework at time step t as follows.

At the beginning of time step t, a surrogate model S_{t-1} is constructed on the solution database $\mathcal{D}_{t-1}$ ($\mathcal{D}_{t-1}$ is a collection of solutions with their fitnesses evaluated at time step $t-1$), the reason of which will be described in Step 1 below. After that, we employ a population-based optimisation algorithm, e.g., PSO (Clerc and Kennedy [2002]), to search for an optimal solution based on the metric in Equation 5.3 when maximizing 'average fitness' or the metric in Equation 5.4 when maximizing 'survival time'. For each candidate solution $\mathbf{x}$ in the population, three steps are performed to calculate the corresponding metric (Note that when maximizing 'survival time', the following 3 steps are performed if $f_t(\mathbf{x}) \geq V$. Otherwise, $f_t(\mathbf{x})$ is returned as the metric.):

- **Step 1: Prepare the solution's fitness time series up to time step t: $(f_{t-L+1}(\mathbf{x}), ..., f_t(\mathbf{x}))$.** We first evaluate $\mathbf{x}$ using the current fitness function f_t to get $f_t(\mathbf{x})$. At time step t, we have previously trained one surrogate model S_{t-i} on each database $\mathcal{D}_{t-i}$, $t-L+1 \leq i \leq t-1$. In practice, we need to maintain only the surrogate models S_{t-i}, $t-L+1 \leq i \leq t-1$, in the memory. Since we model the DFF as a black-box, we cannot evaluate a solution's fitness at previous time steps. Therefore, in order to get a solution's fitness at previous time steps, say $f_{t-j}(\mathbf{x})$, $1 \leq j \leq L-1$ & $1 \leq t-j$, we use the surrogate model S_{t-j} to evaluate $\mathbf{x}$ and return the output as $f_{t-j}(\mathbf{x})$. For more information on how the surrogate model for a previous fitness function is built, readers are referred to Appendix .7.

- **Step 2: Build the ensemble based on the time series $(f_{t-L+1}(\mathbf{x}), ..., f_t(\mathbf{x}))$ produced by Step 1, and predict the solution's future fitness**

using the ensemble. Suppose we are using k learners in the ensemble. For each learner, we train it on the time series data $(f_{t-L+1}(\mathbf{x}), ..., f_t(\mathbf{x}))$. We then make the prediction of each leaner for the solution's fitness at future time steps: $\hat{f}^i_{t+j}(\mathbf{x})$, $1 \leq i \leq k$, for 'average fitness' $1 \leq j \leq T-1$ and for 'survival time' $1 \leq j \leq L^p$. Finally, predictions made by each learner are averaged to give the output of the ensemble, $\hat{f}_{t+j}(\mathbf{x})$, according to Equation 5.2. For more information on how to build each learner on the fitness time series, readers are referred to Appendix .8.

- **Step 3: Calculate the corresponding metric based on the output of Step 2.** Given a solution's predicted future fitness and corresponding prediction standard deviation σ, we are able to calculate the solution's metric in Equation 5.3 when maximizing 'average fitness' or the metric in Equation 5.4 when maximizing 'survival time'.

As our algorithm framework contains components of optimizing and ensemble learning, we term it the framework of Optimizing and Ensemble Learning (OEL). The OEL framework is further summarised in pseudo-code in Algorithm 1. Without loss of generality, we illustrate OEL at time step t. In other words, we assume that the environmental change is known to OEL. Also, there is the computational budget Δe, which is measured in the number of fitness evaluations, for each time step. Note that the time step t starts from 1, and the metrics in Equations 5.3 and 5.4 are not calculated until t is no smaller than L (the fitness time series of length L is required to train the ensemble). When t is smaller than L, the optimiser in OEL searches based on a solution's current fitness.

It should be noted that, although we demonstrate OEL by using PSO as the optimizer, any population-based optimisation algorithm can serve as the optimizer in OEL. The reason is that the population-based optimizer in OEL runs based on the metric calculated entirely outside the optimizer. In other words, with slight change to the pseudo-code between Line 8 and 20 in Algorithm 1, we can apply other population-based optimisation algorithms, such as genetic algorithm, evolution strategy, etc, to OEL.

The general idea of using surrogate models to approximate a solution's previous fitness and learning models to predict a solution's future fitness was inspired by Jin et al.'s work (Jin et al. [2012]). However, OEL differs significantly from (Jin et al. [2012]) in three aspects:

- The metrics in OEL based on which a population-based optimisation algorithm searches for robust solutions are specially designed for the corresponding robustness definition, whereas a single metric that averages a solution's past, current and future predicted fitness was used in Jin et al.'s framework regardless of the corresponding robustness definition.

- Only one global surrogate model is built at each time step, whereas in Jin et al.'s framework (Jin et al. [2012]) every time a solution is evaluated, $(L-1)$ (L is the length of the fitness time series) number of local surrogate models have to be constructed if the current solution has not been evaluated before. The global surrogate model scheme employed in OEL greatly reduces the number of times of training a surrogate model in comparison with Jin et al.'s local surrogate scheme. The factor of reduction can be as high as $\Theta(L * \Delta e)$, where Δe is the number of fitness evaluations at each time step.

- The basic ensemble learning method is used in OEL to predict a solution's future fitness while a single learning model was used in Jin et al.'s framework.

5.2.4 Computational Complexity

Our OEL framework outputs a robust solution at each time step, even though the robust solution found needs not be implemented depending on the robustness definition of 'average fitness' or 'survival time'. In the following, we analyse the computational complexity to arrive at a robust solution at the current time step:

1. **Building a global surrogate model for the last time step:** The global surrogate model employed in our algorithm framework is a RBFN. We use the K-means (MacQueen et al. [1967]) algorithm to first determine the centre of each basis function. Within each cluster, a covariance matrix is estimated for the width of corresponding basis function. A simple least square method is then used to determine the weights for each basis function.

Algorithm 1 Pseudo-code of OEL for ROOT at Time Step t

```
1: Begin
2: if $t == 1$ then
3:     Randomly initialize a population $P_t$, and go to Line 7;
4: end if
5: Randomly initialize a population $P_t$. Replace an individual in $P_t$ with the
   best solution in the solution database $\mathcal{D}_{t-1}$ in terms of fitness $f_{t-1}$;
6: Train a surrogate model $S_{t-1}$ using the solution database $\mathcal{D}_{t-1}$;
7: Create an empty solution database $\mathcal{D}_t$;
8: Repeat
9: for each particle $\mathbf{x}$ in the population $P_t$ do
10:    Evaluate $\mathbf{x}$ using the current fitness function $f_t$. Insert $(\mathbf{x}, f_t(\mathbf{x}), t)$ into the
       solution database $\mathcal{D}_t$, if it is not in $\mathcal{D}_t$;
11:    if $t \geq L$ then
12:        Calculate $\mathbf{x}$'s metric for maximizing 'average fitness' or 'survival time'
           by following the above mentioned three steps, which involve ensemble
           learning;
13:        Update $p_{best}$ (i.e., the best position found by $\mathbf{x}$) and $g_{best}$ (i.e., the best
           position found by the whole swarm) of the swarm based on the metric;
14:    else
15:        Update $p_{best}$ and $g_{best}$ of the swarm based on $f_t(\mathbf{x})$;
16:    end if
17: end for
18: Until $\Delta e$ number of fitness evaluations exhausted
19: Output $g_{best}$ as the best robust solution;
20: End
```

The overall time complexity for building the RBFN is $O(I * n_c * \Delta e * d + \Delta e * d^2 + n_c^3 + n_c * \Delta e * (n_c + d))$, where I is the number of iterations in the K-means algorithm, n_c is the number of basis functions used in RBFN, and d is the number of dimensions in the decision variables. For more details of the RBFN surrogate model, readers are referred to Appendix .7.

2. **Optimizing and learning:** The computational complexity of calculating the metric of a solution is analysed as follows. Firstly, a fitness time series of length L has to be prepared, which incurs a time complexity of $O(L * d * n_c + c_{ef})$. We use c_{ef} to denote the computational cost for one fitness evaluation. Secondly, k number of learners have to be trained on the time series. Three types of learners are employed in the ensemble, i.e., AR (Box et al. [1994]), Nearest Neighbourhood (NN) (Farmer and Sidorowich [1987]), and Moving Average (MA) (Box et al. [1994]). Therefore, the time complexity for training k learners is approximately $O(k * n_f^2 * L)$, where n_f is the order of AR model. We get $O(k * n_f^2 * L)$ because the training time of AR dominates that of NN or MA. Finally, several predictions have to be made using the ensemble, which gives a time complexity of $O(k * T)$ if 'average fitness' is considered or $O(k * L^p)$ if 'survival time' is considered. As a result, the overall time complexity for the population-based search algorithm is $O(\Delta e * (k * n_f^2 * L + c_{ef}))$, considering that the training of each learner for each solution takes most of computational time. For more details of learning models used for time series prediction, readers are referred to Appendix .8.

It is easy to show that the overall time complexity to arrive at a robust solution given Δe number of fitness evaluations at each time step is donated by the K-means algorithm in RBFN training, the ensemble model training for time series prediction, and fitness evaluations, which is therefore $O(\Delta e * (I * n_c * d + k * n_f^2 * L + c_{ef}))$. It should be noted that OEL does not incur any additional fitness evaluation in evaluating the corresponding metric of a solution except for the fitness evaluation of the solution's current fitness.

5.3 Experimental Studies

The first goal of the experimental studies is to show the effectiveness of our OEL framework compared to existing EDO approaches on ROOT problems. In other words, we would like to answer the question that under what type of DFFs it is better to use other EDO approaches over OEL for ROOT problems and vice versa. The second goal of the experimental studies is to investigate whether the ensemble learning mechanism offers an improvement over single model learning in solving ROOT problems.

5.3.1 Experimental Settings

5.3.1.1 Test Problems

There have been a number of benchmarks for constructing DFFs (Branke [1999], Morrison and De Jong [1999], Jin and Sendhoff [2004], Li et al. [2008]). The common idea of these benchmarks is to first define a baseline fitness function with some configurable parameters. The dynamics are then defined to change one fitness function to another by updating those configurable parameters. The first baseline fitness function used in the experimental studies takes the form as follows:

$$f_t(\mathbf{x}) = \max_{i=1}^{i=m}\{h_t^i - w_t^i * ||\mathbf{x} - \mathbf{c}_t^i||_2\}. \tag{5.6}$$

The baseline fitness function in Equation 5.6 is first defined in (Morrison and De Jong [1999]), where scalars h_t^i, w_t^i, and vector $\mathbf{c}_t^i, \mathbf{c}_t^i = (c_1^i, ..., c_d^i)$, denote the height, the width, and the centre of the ith peak function at time step t. $\mathbf{x}$ is the vector of decision variables. m is the total number of peaks, and d is the number of dimensions. The cone-like baseline fitness function in Equation 5.6 is by far the most commonly used one in DFF benchmarks (Branke [1999], Morrison and De Jong [1999], Nguyen et al. [2012a]). Therefore, in order to represent a vast majority of DFF benchmarks studied in the literature, we employ the cone-like baseline fitness function defined in Equation 5.6 as the first baseline fitness function.

As discussed in Chapter 4, it is not sufficient to test an algorithm's ROOT ability on existing DFF benchmarks as it is difficult to know the absolute best performance on existing DFF benchmarks. We therefore include the two baseline fitness functions developed in Chapter 4 in the experimental studies. The baseline fitness function for maximizing 'average fitness' is:

$$f_t(\mathbf{x}) = \frac{1}{d}\sum_{j=1}^{d} \max_{i=1}^{i=m}\{h_t^{ij} - w_t^{ij} * |x_j - c_t^{ij}|\}\,, \tag{5.7}$$

and the baseline fitness function for maximizing 'survival time' is:

$$f_t(\mathbf{x}) = \min_{j=1}^{j=d}\{\max_{i=1}^{i=m}\{h_t^{ij} - w_t^{ij} * |x_j - c_t^{ij}|\}\}\,, \tag{5.8}$$

where, in both baseline fitness functions, scalars h_t^{ij}, w_t^{ij}, and c_t^{ij} denote the height, the width, and the centre of the ith peak function for the jth dimension at time step t. $\mathbf{x}$ represents the decision variables with d dimensions ($\mathbf{x} = (x_1, ..., x_d)$), and m is the number of peaks for each dimension.

In order to answer the question that under what type of DFFs it is better to use existing approaches over OEL for ROOT problems and vice versa, The 6 different dynamics suggested in the dynamic benchmark generator (Li et al. [2008]), which have been described in Section 4.2.2, are applied to change the baseline fitness functions in Equations 5.6, 5.7, and 5.8.

Applying those 6 different types of dynamics and the rotation technique to the three baseline fitness functions in Equations 5.6, 5.7, and 5.8, we obtain 18 different DFFs. We refer them as TP_{ij}, $i = 1, 2, 3$ & $j = 1, 2, ..., 6$, where TP_{ij} indicates the benchmark that is generated by applying the jth dynamic to the ith baseline fitness function. TP_{1j}, $j = 1, 2, ..., 6$, are used for maximizing both 'average fitness' and 'survival time', while TP_{2j}, $j = 1, 2, ..., 6$, are used only for maximizing 'average fitness', and TP_{3j}, $j = 1, 2, ..., 6$, are used only for maximizing 'survival time'. For each TP_{ij}, we generate 200 consecutive fitness functions as follows. For the first fitness function, we randomly initialize the height and the width in their corresponding ranges, and the centre across the solution space. To generate the next fitness function, we apply the jth dynamic to the current height, width, and rotation angle. The centre of the next fitness function is obtained by rotating the centre of the current fitness function using the updated rotation angle. Whenever the height, width, or centre gets out of their corresponding ranges, we reset them to their upper limits (if larger than upper limits) or lower limits (if smaller than lower limits). Note the exceptions that for the *chaotic* change the centre is updated dimension by dimension using Equation 4.11 rather than using the rotation technique, and for the *recurrent* change and the *recurrent with noise* change the rotation angle θ is fixed to $\frac{2\pi}{P}$, where P is the period and the centre is rotated following a fixed direction. A summary of all test problem parameters are presented in Table 5.1. In the experimental studies in Chapter 4, we set the time window T to 2, 6, and 10, which cover a relatively large range. In this chapter in Table 5.1, we set the time window T to 2, 3, and 4. The reason is that we would also like to evaluate the performance of maximizing 'average fitness' in ROOT when it is not required to predict far into the future. Yet, experimental results are still shown on large time windows on one problem instance in Figure 5.7.

Table 5.1: Parameter settings of the test problems

number of peaks m	5
number of dimensions d	2
search range	$[-25, 25]$
height range	$[30, 70]$
width range	$[1, 13]$
angle range	$[-\pi, \pi]$
$height_{severity}$	5.0
$width_{severity}$	0.5
$angle_{severity}$	1.0
initial angle θ	0
γ	0.04
γ_{max}	0.1
chaotic constant A	3.67
period P	12
$noise_{severity}$	0.8
number of dimensions l for rotation	2
time window T	2,3,4
fitness threshold V	40,45,50
computational resource for each time step Δe	2500

5.3.1.2 Performance Measurement

We use the performance measurement defined in Equation 4.18 in Section 4.3.1.3, and the reason is the same as that explained in Section 4.3.1.3. Given a sequence of fitness functions $(f_1, f_2, ...f_N)$, the performance measurement is rewritten in the following for clarity:

$$Performance^{ROOT} = \frac{1}{N}\sum_{i=1}^{N} F(\mathbf{x}_i), \tag{5.9}$$

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

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

5.3.1.3 The Setting of PSO

We adopt a simple PSO as the optimizer in OEL. The PSO follows the constriction version (Clerc and Kennedy [2002]). The swarm population size is 50. The constants c_1 and c_2, which are used to bias a particle's attraction to the personal best and the global best, are both set to 2.05, and hence the constriction factor χ takes a value of 0.729844. The velocity of particles is constricted within the range $[-v_{max}, v_{max}]$. The value of v_{max} is set to the range of the search space, which is 50 in our case.

5.3.1.4 Learning Models for Approximation and Prediction

The global surrogate model employed is RBFN because of its relatively lower time complexity compared with other non-linear surrogate models. For the RBFN, the K-means algorithm is first used to determine the centre of each basis function, where 500 iterations are executed. The number of basis functions, i.e., the number of clusters in the K-means algorithm, is arbitrarily fixed to 10. Normalised

Gaussian radial basis functions are used, and a density estimation technique is used to determine the width for each basis function. Finally, a least square solution is obtained for the weight of each basis function. For more information about the surrogate model, readers are referred to Appendix .7.

The time series prediction models are AR, NN, and MA, which serve as the 3 learners in the ensemble in OEL. The reasons for using AR, NN, and MA are two-fold. Firstly, they all have low time complexity so a solution's metric can be calculated efficiently. Secondly, they possess different characteristics in the sense that AR is a linear model in the input space; NN is good at prediction in recurrent DFFs; and MA is appropriate for DFFs which change randomly. In other words, AR, NN, and MA are more likely to have mutually independent prediction error than, e.g., 3 AR models which differ only in the embedding size. The embedding size for the time series in AR is set to 4, i.e., the order of AR is 4. The coefficients of AR are determined using the least square technique. The embedding size for the time series in NN is also 4, and the size of neighbourhood in NN is 3. For MA, the average of the latest 3 data points in the time series is used to predict future data points. For more details of learning models for time series prediction, readers are referred to Appendix .8.

$T-1$ steps into the future are predicted when maximizing 'average fitness' with time window T. Four, i.e., $L^p = 4$, steps into the future are predicted when maximizing 'survival time'.

5.3.1.5 Methods for Comparison

We select three typical approaches from the literature for DFFs to compare their effectiveness on ROOT problems with OEL. The first approach is a simple PSO algorithm with a re-start strategy, which we denote as 'RPSO' hereafter. The re-start strategy means the best solution found for the last fitness function is copied into the initial population for the current fitness function, and all other particles are initialized randomly. The corresponding PSO in 'RPSO' is the same as the optimizer we employed in OEL. The second approach can be seen as an ideal TMO approach, where at time step t a solution that maximizes f_t is chosen[1] as the

[1]No computation is involved in selecting a solution that maximizes the fitness function f_t in our benchmarks since we know beforehand which solution maximizes f_t in our benchmarks.

ROOT solution at time step t. The ideal TMO approach, denoted as 'optimum' hereafter, can be viewed as the best any TMO approach can do for TMO. Note that the performance of 'optimum' is not necessarily better than 'RPSO' on ROOT problems since both are designed for TMO instead of ROOT. The third approach is Jin et al.'s framework (Jin et al. [2012]) designed specially for ROOT problems. Jin et al.'s framework, denoted as 'Jin's' hereafter, is implemented as follows. The same surrogate model RBFN for estimating a solution's past fitness is used in 'Jin's' as in OEL. The AR model with order 4 is used for time series prediction in 'Jin's'. One previous estimated fitness and four future predicted fitnesses are used in the metric in 'Jin's', the setting of which is reported to have the best performance in (Jin et al. [2012]). For more details of 'Jin's', readers are referred to (Jin et al. [2012]).

The three methods 'RPSO', 'optimum', and 'Jin's' have also been selected in the experimental studies in Chapter 4, and they are selected in this Chapter for the same reasons, which have been discussed in Section 4.3.1.2.

In order to justify the ensemble learning mechanism in OEL in comparison with single model learning for ROOT problems, we create other versions of OEL, in which only one learner is used for time series prediction in ROOT. Since the ensemble investigated in OEL includes three learning models, i.e., AR, NN, and MA, we create 3 other versions of OEL by setting the time series prediction model to be only AR, NN, and MA respectively. Without any ambiguity, we will denote OEL as the method which uses ensemble learning for time series prediction, and those three other versions of OEL as 'AR', 'NN', and 'MA', respectively. The settings of AR model in 'AR', NN model in 'NN', and MA model in 'MA' are the same as those in OEL.

It should be noted that all the methods in comparison on the performance of ROOT are given the same number of fitness evaluations, i.e., $\Delta e = 2500$, at each time step.

5.3.2 Simulation Results

It should be noted that all the results presented below are for time steps L (the length of fitness time series) to 100 since the prediction of a solution's future fitness starts when there are L number of data points available. L is set to 20, except for the results presented in Figures 5.5 and 5.6, where we examine the influence of L on the performance of OEL. The maximal 'survival time' is set to 10 as some solution's 'survival time' can be infinite in certain benchmarks. The results in Tables 5.2, 5.3, 5.4, and 5.5 are averaged over 30 independent runs. Results with a '+' or '-' are significantly better or worse than those of OEL at the 0.05 significance level using the Wilcoxon rank sum test[1].

5.3.2.1 Peer Methods Versus OEL

The results for maximizing 'average fitness' using 'RPSO', 'Optimum', 'Jin's', and OEL are presented in Table 5.2. We can see that OEL significantly outperforms other methods for ROOT problems in 12 benchmark problems with 6 different dynamics with few exceptions. On test problems TP_{13} and TP_{23}, 'RPSO' and 'Optimum' outperform OEL. The reason is that in TP_{13} and TP_{23} the DFFs change entirely at random. Learning in this case would mostly be fitting noises and can mislead the optimizer in OEL. In other words, for DFFs which change entirely at random, TMO is equivalent to maximizing 'average fitness' in ROOT. To account for the poor performance of OEL in TP_{25} and TP_{26} when T is 2, the reasons can be two-fold. Firstly, the time window is short as compared to $T = 3$ and $T = 4$, and thus good solutions for the current time step can still have a good 'average fitness'. Secondly, the fitness landscapes of TP_{25} and TP_{26} are more rugged than others, and therefore the prediction accuracy of the surrogate in OEL is lower, which presents large noise in the fitness time series.

In comparison with 'Jin's', OEL outperforms it in nearly all cases with the exceptions of TP_{11} and TP_{15} when $T = 4$. This may be due to the fact that, when maximizing 'average fitness' with $T = 4$, the metric in 'Jin's' takes a similar form to OEL's metric in Equation 5.3. The reasons why OEL outperforms 'Jin's' in all other cases are as follows. On the one hand, the metric used in OEL for maximizing 'average fitness' is adapted with the time window T while a fixed number (4 investigated in our experiment) of predicted future fitnesses, a fixed number (1 investigated in our experiment) of estimated previous fitness, together

[1]The p-values have been adjusted using Holm-Bonferroni corrections (Holm [1979]) for each column in the Table over 12 test problems.

Table 5.2: The averaged performance measure in Equation 5.9 over 30 runs under the robustness definition of 'average fitness' with different settings of time window T (peer methods versus OEL).

Test	RPSO			Optimum			Jin's			OEL		
Problem	$T=2$	$T=3$	$T=4$	$T=2$	$T=3$	$T=4$	$T=2$	$T=3$	$T=4$	$T=2$	$T=3$	$T=4$
TP_{11}	48.19-	42.24-	39.33-	48.75-	43.16-	40.49-	49.01-	46.06-	44.97	49.45	46.67	44.92
TP_{21}	51.55-	50.41-	48.33-	51.95-	50.69-	48.23-	51.52-	50.78-	49.72-	52.60	51.50	50.01
TP_{12}	26.64-	17.42-	11.46-	26.87-	17.87-	12.33-	26.54-	22.89-	20.52-	32.75	26.04	22.56
TP_{22}	44.81	36.89-	33.95-	45.29+	37.13-	34.36-	40.29-	36.30-	34.28-	44.83	38.48	35.44
TP_{13}	38.75	32.36+	26.98+	39.02+	31.91+	27.05+	29.99-	26.79-	24.50-	38.78	31.47	26.21
TP_{23}	48.85+	45.18+	42.73+	48.29+	43.84-	41.12-	43.73-	41.63-	40.34-	47.95	44.35	41.75
TP_{14}	-0.28-	6.31-	-11.79-	0.10-	5.85-	-11.87-	4.63-	3.23-	-0.80-	12.88	12.99	2.71
TP_{24}	48.15-	43.36-	40.11-	48.89-	43.72-	40.17-	45.93-	43.70-	42.20-	49.45	45.04	42.99
TP_{15}	48.17-	33.56-	22.00-	44.92-	27.87-	16.47-	48.22-	34.47-	24.22	49.49	35.49	24.55
TP_{25}	52.34+	46.09-	40.30-	52.58+	46.33	40.99-	49.04-	44.49-	40.69-	51.37	46.31	42.03
TP_{16}	54.04-	48.11-	43.74-	49.73-	43.12-	38.87-	51.86-	50.40-	48.96-	55.12	51.60	49.61
TP_{26}	56.78+	52.28	48.92-	56.45-	52.08-	48.76-	55.09-	51.41-	48.22-	56.59	52.23	49.77

with a solution's current fitness are used to calculate the metric in 'Jin's'. This is rather inflexible as in different situations users would like a solution to be robust for different length of time. More importantly, this inflexibility leads to the larger gap between 'Jin's' and OEL when $T = 2$ than when $T = 4$. Furthermore, only one learning model (AR) is used for the prediction task in 'Jin's', whereas the basic ensemble learning method is used for the prediction of a solution's future fitness in OEL. The improved prediction accuracy due to ensemble learning in OEL further escalates OEL's performance over that of 'Jin's'.

Table 5.3 presents the results of maximizing 'survival time' using 'RPSO', 'Optimum', 'Jin's', and OEL. We can observe that OEL outperforms 'RPSO' and 'Optimum' with few exceptions. Again in randomly changing DFFs, i.e., TP_{32} and TP_{33}, 'RPSO' and 'Optimum' sometimes outperform OEL when V is 45 and 50. This might be caused by large random changes in DFFs, which make OEL's learning ineffective. In some other cases, like TP_{34} and TP_{15}, where OEL performs worsen than 'RPSO' and 'Optimum', this is also due to OEL's poor prediction accuracy. The reason why OEL did not predict well in those DFFs needs to be further investigated. In all other cases where OEL outperforms 'RPSO' and 'Optimum', the ensemble learning model provides useful information in quantifying a solution's future expected fitness and fitness variance, and also the metric in Equation 5.4 for maximizing 'survival time' in OEL guides the optimizer to converge to robust solutions in terms of 'survival time'.

Table 5.3: The averaged performance measure in Equation 5.9 over 30 runs under the robustness definition of 'survival time' with different settings of fitness threshold V (peer methods versus OEL).

Test	RPSO			Optimum			Jin's			OEL		
Problem	$V = 40$	$V = 45$	$V = 50$	$V = 40$	$V = 45$	$V = 50$	$V = 40$	$V = 45$	$V = 50$	$V = 40$	$V = 45$	$V = 50$
TP_{11}	1.99-	1.75-	1.43-	2.28-	2.01-	1.51-	3.23-	2.47-	1.52-	3.71	2.60	1.80
TP_{31}	2.11-	1.61-	1.13-	1.94-	1.48-	1.11-	3.22-	1.86-	0.91-	3.53	2.39	1.26
TP_{12}	1.21-	1.13-	1.05-	1.22-	1.15-	1.07	0.76-	0.63-	0.45-	1.31	1.20	1.07
TP_{32}	1.10-	1.02	0.97+	1.11	1.04+	0.96	0.76-	0.57-	0.46-	1.13	1.02	0.95
TP_{13}	1.23	1.08-	1.00-	1.17-	1.07-	1.04	0.74-	0.57-	0.43-	1.25	1.13	1.04
TP_{33}	1.31	1.12-	0.98	1.28	1.09-	1.02+	1.00-	0.72-	0.50-	1.31	1.19	1.00
TP_{14}	1.00-	1.00-	1.00	1.00-	1.00-	1.00	0.39-	0.31-	0.22-	1.03	1.01	1.00
TP_{34}	1.26-	1.13-	1.08	1.38	1.21	1.12+	1.09-	0.83-	0.59-	1.40	1.21	1.08
TP_{15}	2.51+	2.22+	1.48-	2.14-	1.79-	1.35-	2.45+	1.99-	1.48-	2.33	2.07	1.64
TP_{35}	1.34-	0.95-	0.92-	1.33-	0.83-	0.83-	1.10-	0.87-	0.72-	1.38	0.99	0.93
TP_{16}	3.16-	2.47-	1.88-	2.26-	1.90-	1.43-	2.92-	2.34-	1.70-	3.30	2.64	1.92
TP_{36}	2.34-	1.75-	1.32-	2.25-	1.53-	1.19-	2.25-	1.75-	1.31-	2.61	2.10	1.53

According to Table 5.3, OEL outperforms 'Jin's' in nearly all cases with exception for TP_{15} when $V = 40$. The reasons for the success of OEL over 'Jin's' can be summarised as follows. The metric used in 'Jin's' is simply an average of a solution's estimated past, evaluated current, and predicted future fitness, which does not care about whether the current fitness is above the threshold V. This can be problematic as a solution can have a very poor current fitness even though it has a very good metric value in 'Jin's'. According to the 'survival time' definition in Equation 3.3 in Chapter 3, a solution's 'survival time' is 0 if its current fitness is below the fitness threshold V. Also, the metric in 'Jin's' is fixed no matter what the corresponding robustness definition is in ROOT. When it comes to maximizing 'survival time', not only a solution's future expected fitness should be considered but also its future fitness variance. The metric in Equation 5.4 in OEL calculates a solution's expected 'survival time' directly, which guides the optimizer to robust solutions in terms of 'survival time'. However, in the metric of 'Jin's', no information is provided about a solution's expected 'survival time'.

5.3.2.2 Single Model Versus OEL

The results of maximizing 'average fitness' using a single learner for time series prediction and OEL are presented in Table 5.4. We can see that OEL generally outperforms 'AR', 'NN', and 'MA'. In cases where OEL is not the best, OEL always comes as the second best. The results confirm our claim that the ensemble learning mechanism in OEL reduces the MSE in predicting a solution's future fitness, which leads to better robust solutions in terms of 'average fitness'. This is further illustrated in Figure 5.1, where the Root-Mean-Square Error (RMSE) at each time step is presented for methods 'AR', 'NN', 'MA', and OEL for TP_{14} with T being 2. The RMSE for time step t is calculated as $\sqrt{\frac{1}{\Delta e}\sum_{i=1}^{\Delta e}(f_{t+1}(\mathbf{x}_i) - \hat{f}_{t+1}(\mathbf{x}_i))^2}$, where Δe number of solutions have been evaluated during time step t. $f_{t+1}(\mathbf{x}_i)$ is $\mathbf{x}_i$'s true fitness at time step $t+1$, and $\hat{f}_{t+1}(\mathbf{x}_i)$ is the predicted fitness at time step $t+1$. We can see that the RMSE is reduced in OEL in comparison with 'AR', 'NN', and 'MA'. Similar phenomena can be observed in other cases. In cases where 'NN' outperforms OEL, they are mainly on DFFs which change periodically over time, such as TP_{15}, TP_{25}, and TP_{26}. This is because the NN learning model is more suitable for the prediction of periodic DFFs than AR and MA learning models. As for the case where 'MA' outperforms OEL in TP_{23}, this is because in TP_{23} the DFF changes totally at random, and a MA learning model is more appropriate than AR and NN learning models in this case.

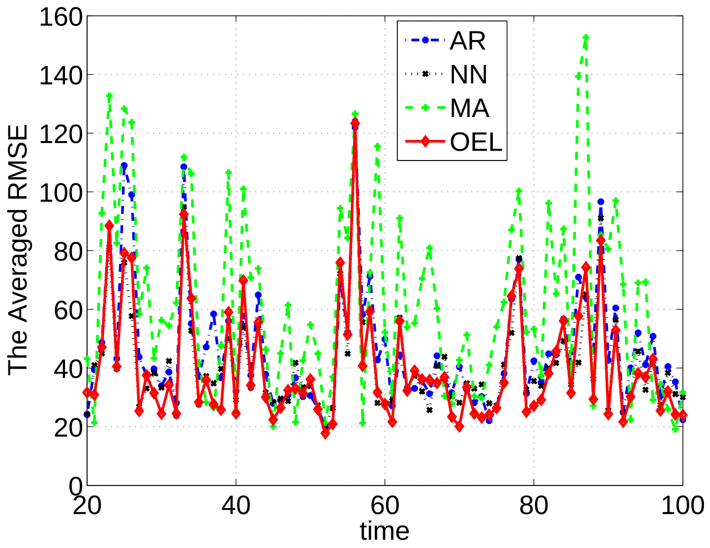


Figure 5.1: The averaged RMSE over 30 runs at each time step obtained by methods 'AR', 'NN', 'MA', and OEL on TP_{14} with T being 2.

The results of maximizing 'survival time' using single learners for time series prediction and OEL are presented in Table 5.5. The performance of OEL is

Table 5.4: The averaged performance measure in Equation 5.9 over 30 runs under robustness definition of 'average fitness' with different settings of time window T (single model versus OEL).

Test	AR			NN			MA			OEL		
Problem	$T=2$	$T=3$	$T=4$	$T=2$	$T=3$	$T=4$	$T=2$	$T=3$	$T=4$	$T=2$	$T=3$	$T=4$
TP_{11}	49.02-	45.98-	44.32-	48.65-	46.16-	43.95-	50.15+	46.85	44.54-	49.45	46.67	44.92
TP_{21}	52.32-	51.29-	50.02	52.01-	50.69-	49.08-	52.27-	51.00-	49.42-	52.60	51.50	50.01
TP_{12}	31.64-	25.08-	21.21-	30.70-	23.71-	21.04-	32.34	25.23-	20.62-	32.75	26.04	22.56
TP_{22}	44.22-	37.52-	35.17	43.82-	36.93-	34.31-	44.55	38.07	35.15-	44.83	38.48	35.44
TP_{13}	37.73-	31.37	26.74+	36.96-	30.16-	26.08	38.67	31.01-	25.79-	38.78	31.47	26.21
TP_{23}	47.30-	43.44-	40.68-	46.67-	42.91-	40.65-	48.20+	44.72+	42.33+	47.95	44.35	41.75
TP_{14}	9.76-	9.14-	-0.18-	10.58-	10.02-	0.05-	9.50-	12.43	0.36-	12.88	12.99	2.71
TP_{24}	49.17	45.00	42.89	48.79-	44.11-	42.03-	48.48-	43.60-	41.36-	49.45	45.04	42.99
TP_{15}	49.01-	34.58-	23.79-	48.93-	36.05+	25.12	47.38-	32.56-	22.45-	49.49	35.49	24.55
TP_{25}	51.19	46.03	41.06-	51.59+	46.38	43.24+	49.36-	42.83-	38.17-	51.37	46.31	42.03
TP_{16}	51.20-	47.29-	44.53-	54.16-	50.69-	48.27-	54.14-	48.89-	45.74-	55.12	51.60	49.61
TP_{26}	56.19-	51.79-	48.34-	56.85+	52.87+	51.09+	56.13-	51.35-	48.10-	56.59	52.23	49.77

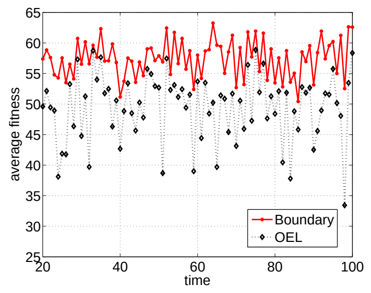
Table 5.5: The averaged performance measure in Equation 5.9 over 30 runs under robustness definition of 'survival time' with different settings of fitness threshold V (single model versus OEL).

Test	AR			NN			MA			OEL		
Problem	$V=40$	$V=45$	$V=50$	$V=40$	$V=45$	$V=50$	$V=40$	$V=45$	$V=50$	$V=40$	$V=45$	$V=50$
TP_{11}	3.53	2.62	1.76	3.60	2.57	1.80	3.15-	2.26-	1.69-	3.71	2.60	1.80
TP_{31}	3.49	2.28	1.26	3.50	2.33	1.25	2.91-	2.14-	1.24	3.53	2.39	1.26
TP_{12}	1.30	1.20	1.06	1.27-	1.18	1.05	1.24-	1.13-	1.06	1.31	1.20	1.07
TP_{32}	1.12	1.02	0.95	1.14	1.03	0.96	1.11	1.02	0.96	1.13	1.02	0.95
TP_{13}	1.24	1.15	1.04	1.27	1.15	1.03	1.21-	1.10-	0.99-	1.25	1.13	1.04
TP_{33}	1.33	1.18	1.00	1.28	1.17	0.99	1.36+	1.20	0.99	1.31	1.19	1.00
TP_{14}	1.04	1.00	1.00	1.03	1.01	1.00	1.02-	1.00-	1.00	1.03	1.01	1.00
TP_{34}	1.39	1.19	1.09	1.34-	1.19	1.08	1.34-	1.17-	1.08	1.40	1.21	1.08
TP_{15}	2.41+	2.06	1.55-	2.42+	2.28+	1.91+	2.08-	1.85-	1.40-	2.33	2.07	1.64
TP_{35}	1.35	0.95-	0.91-	1.43+	1.03+	0.94	1.19-	0.93-	0.90-	1.38	0.99	0.93
TP_{16}	3.12-	2.53-	1.82-	3.38	2.63	1.97	2.93-	2.47-	1.83-	3.30	2.64	1.92
TP_{36}	2.57	1.87-	1.37-	2.67	2.18	1.59	2.22-	1.73-	1.29-	2.61	2.10	1.53

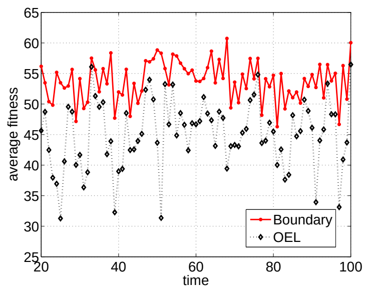
slightly worse than that of 'NN' and better than those of 'AR' and 'MA'. There does not seem to be any substantial differences between different learning models. 'MA' seems to perform worse than OEL in most cases, and 'AR' and 'NN' seem to be comparable to OEL in general. There are only two cases, TP_{15} and TP_{35}, where 'NN' performs better than OEL. This is again due to that NN model is more suitable for the prediction of periodic DFFs than models AR and MA.

5.3.2.3 The Absolute Best Performance Versus OEL

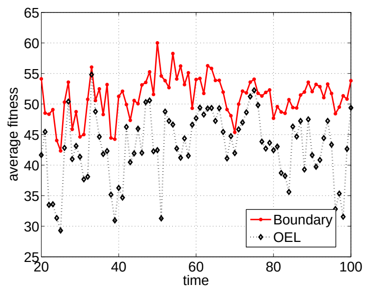
The absolute best performance means the best performance on a certain benchmark. At a certain time step, the absolute best performance, denoted as 'Boundary', is the best robustness (i.e., 'average fitness' or 'survival time') any solution in the solution space can have. The averaged results obtained by OEL over 30 runs at each time step and the absolute best performance in benchmark TP_{24} are plotted in Figure 5.2. From Figure 5.2, we can see that the difference between the two generally gets larger as T increases when maximizing 'average fitness', which indicates that it is more difficult to predict distant future.



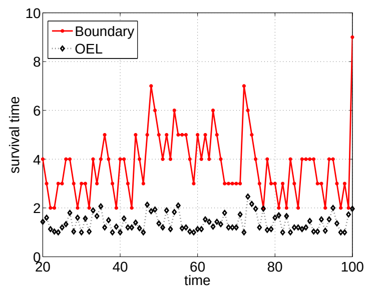
(a) Time window $T = 2$



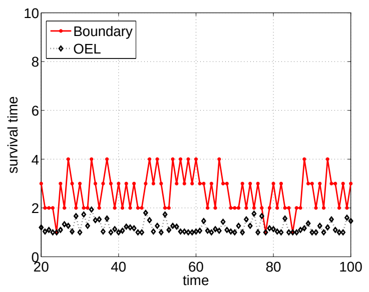
(b) Time window $T = 3$



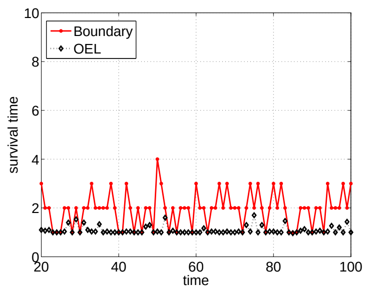
(c) Time window $T = 4$



(d) Fitness threshold $V = 40$



(e) Fitness threshold $V = 45$



(f) Fitness threshold $V = 50$

Figure 5.2: The absolute best performance of the robustness 'average fitness' in benchmark TP_{24} and the robustness 'survival time' in benchmark TP_{34} with different settings of T and V respectively, compared to results obtained by OEL.

For the difference shown in maximizing 'survival time' in Figure 5.2, the gap, in other words the difficulty in maximizing 'survival time', increases as the fitness threshold V decreases. The reasons may be as follows. On the one hand, maximizing a solution's 'survival time' involves two steps. The first step is to maximize the solution's current fitness till V, and the second step is to maximize the number of consecutive future time steps during which the solution's fitness is no smaller than V at each time step. On the other hand, it is relatively easy to maximize the current fitness function in benchmarks employed in this chapter, which can be verified noticing that the averaged solution's 'survival time' at each time step in Figure 5.2 is at least 1. As a result, it may be more difficult maximizing a solution's future fitness than maximizing the solution's current fitness till V in the benchmarks employed in this chapter. In other words, it is harder to find a best solution in terms of 'survival time' with a smaller V in our experimental studies because the solution space with fitness larger than V increases as V decreases. However, it should be noted that in other situations, where it is difficult to maximize a solution's current fitness, the difficulty in maximizing 'survival time' may decrease as V decreases.

5.3.3 More Discussions on OEL

In this subsection, we investigate the influence of some important parameters in OEL. Moreover, the scalability of OEL for ROOT problems is examined by varying the time window T when maximizing 'average fitness' and the fitness threshold V when maximizing 'survival time'.

5.3.3.1 Varying the Number of Clusters in the K-means in OEL

The surrogate model for estimating a solution's previous fitness plays an important role in OEL as it provides the data based on which a solution's future fitness is predicted. We employed a RBFN as the surrogate model in OEL, and we have previously fixed the number of clusters in the K-means in the RBFN to 10. Additional experiments are conducted with regard to the sensitivity of the performance of OEL on the number of clusters in the K-means, the results of which are presented in Figures 5.3 and 5.4.

From Figures 5.3 and 5.4, we can see that the performance of OEL is not influenced much by the number of clusters in the K-means.

5.3.3.2 Varying the Length of Fitness Time Series in OEL

A solution's future fitness is predicted by solving a time series prediction problem via ensemble learning in OEL. An important parameter with regard to the fitness time series is the length L of the time series. The parameter L has been previously fixed to 20. Additional experiments are conducted with regard to the effects of

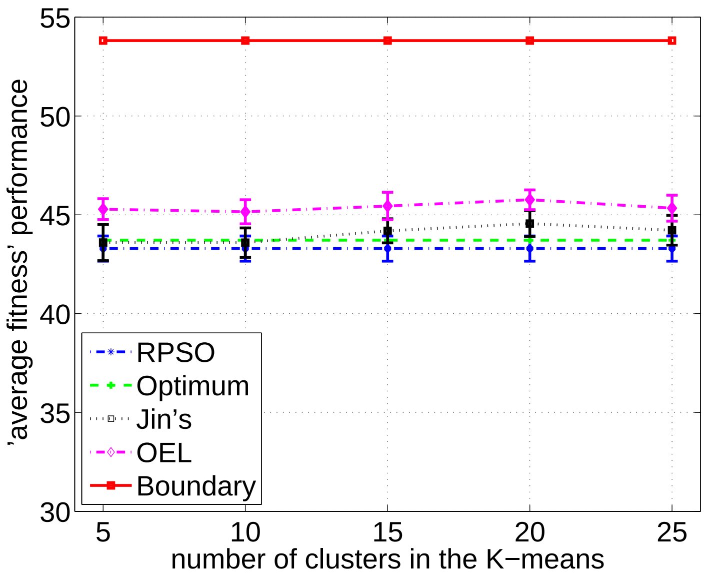


Figure 5.3: The averaged performance measure in Equation 5.9 over 30 runs with one standard deviation errorbar under different settings of number of clusters in the K-means when maximizing 'average fitness' with $T = 3$ on TP_{24}.

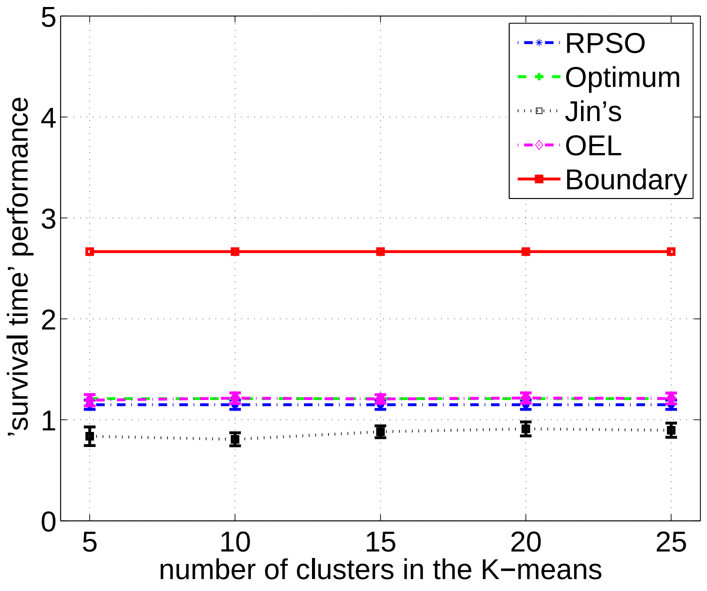


Figure 5.4: The averaged performance measure in Equation 5.9 over 30 runs with one standard deviation errorbar under different settings of number of clusters in the K-means when maximizing 'survival time' with $V = 45$ on TP_{34}.

L on the performance of OEL, the results of which are presented in Figures 5.5 and 5.6.

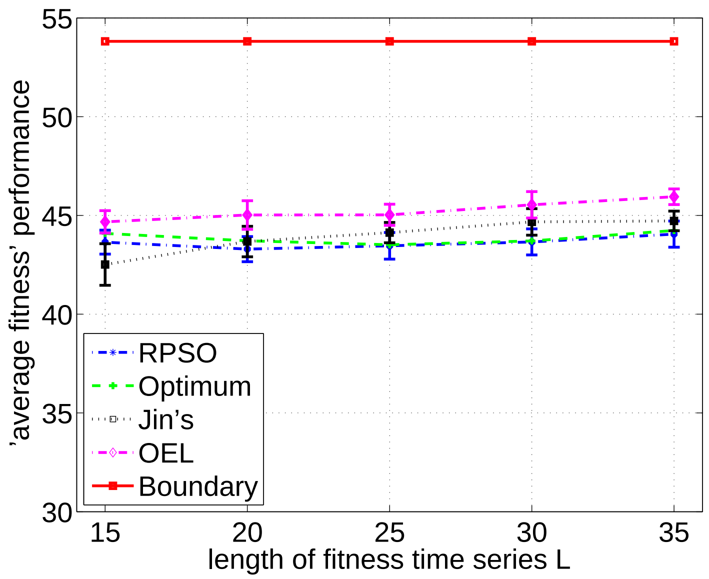


Figure 5.5: The averaged performance measure in Equation 5.9 over 30 runs with one standard deviation errorbar under different settings of length of the fitness time series when maximizing 'average fitness' with $T = 3$ on TP_{24}.

In general, the performance of OEL when maximizing 'average fitness' improves as the time series length L increases since more data is used to predict a solution's future fitness. The observation that the performance of OEL when maximizing 'survival time' does not improve as L increases may indicate the relatively poor performance of OEL when maximizing 'survival time'.

5.3.3.3 Varying the Problem Parameter T in ROOT

When maximizing 'average fitness' in ROOT, the problem parameter T is the number of consecutive time steps based on which a solution's 'average fitness' is calculated. A larger T means that a solution is used for more consecutive time steps. Additional experiments are conducted by varying T from 2 to 10 in order to examine the performance of OEL in a wider range of T, the results of which are presented in Figure 5.7.

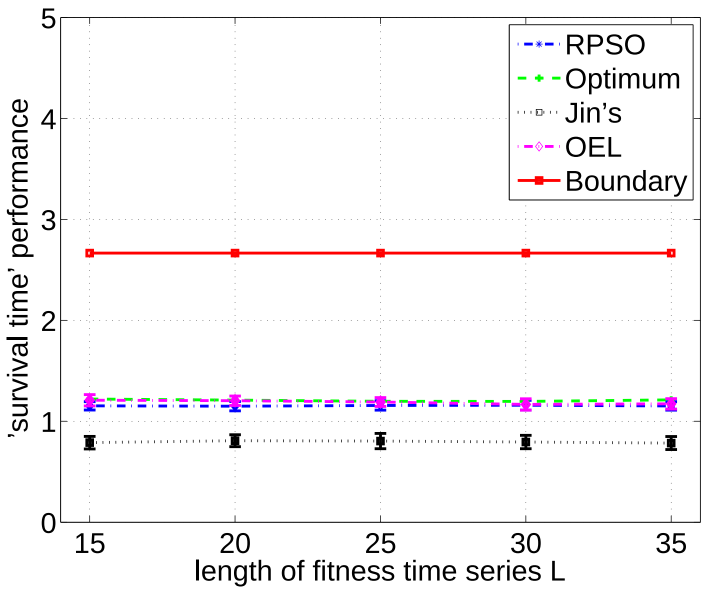


Figure 5.6: The averaged performance measure in Equation 5.9 over 30 runs with one standard deviation errorbar under different settings of length of the fitness time series when maximizing 'survival time' with $V = 45$ on TP_{34}.

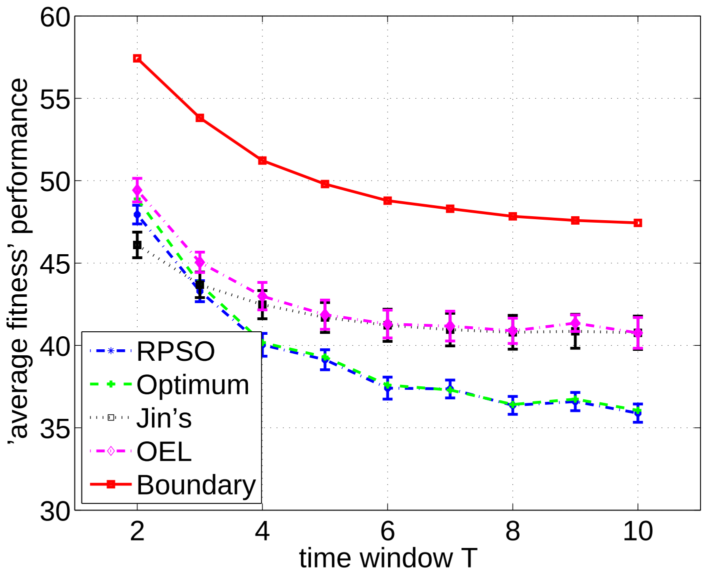


Figure 5.7: The averaged performance measure in Equation 5.9 over 30 runs with one standard deviation errorbar under different settings of T when maximizing 'average fitness' on TP_{24}.

From Figure 5.7, we can see that OEL performs best under different settings of T. The performance of 'Jin's' is comparable to OEL when T is large. Also, the gap between the performances of TMO methods (i.e., 'RPSO' and 'Optimum') and those of OEL and 'Jin's' gets larger as T increases.

5.3.3.4 Varying the Problem Parameter V in ROOT

The problem parameter V is the fitness threshold that indicates the minimal fitness a solution is required to have in order to be robust when maximizing 'survival time' in ROOT. A larger V means that a smaller portion of solutions in the solution space will have a fitness no smaller than V at a time step. Additional experiments are conducted by varying V from 40 to 50 in order to examine the performance of OEL in a wider range of V, the results of which are presented in Figure 5.8.

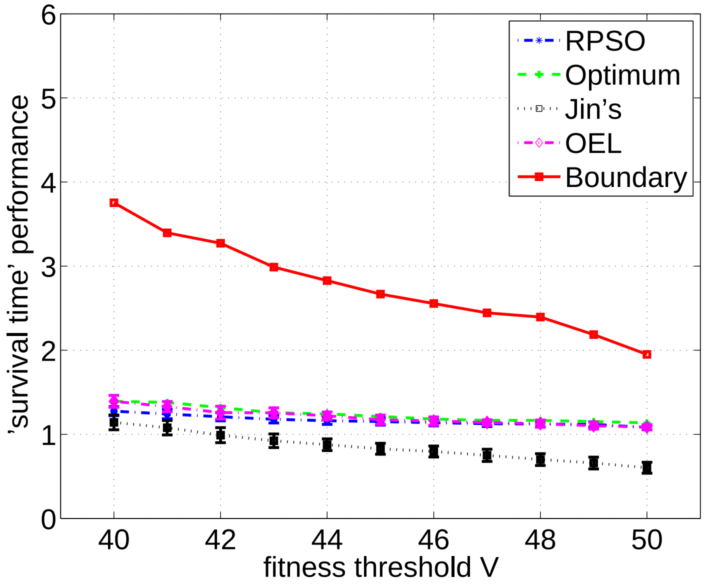


Figure 5.8: The averaged performance measure in Equation 5.9 over 30 runs with one standard deviation errorbar under different settings of V when maximizing 'survival time' on TP_{34}.

From Figure 5.8, we can see that the performance of OEL is better than 'Jin's'. Yet, the performance of OEL is comparable to 'RPSO' and 'Optimum', and future work is needed to improve OEL's performance in maximizing 'survival time' in ROOT.

5.4 Summary

The major contribution of this chapter is a new algorithm framework, i.e., OEL, for ROOT problems. More specially, two new metrics in Equations 5.3 and 5.4 were developed, which are used to guide population-based search algorithms towards optimal solutions in terms of 'average fitness' and 'survival time' respectively in ROOT. Secondly, we proposed to estimate a solution's previous fitness in ROOT via building a global surrogate model, which greatly reduces the time complexity compared to existing strategies. Finally, we proposed to predict a solution's future fitness in ROOT via ensemble learning compared to existing single model learning strategies. Ensemble learning, compared to single model learning, improves the prediction accuracy of a solution's future fitness in ROOT and eventually the quality of solutions to ROOT problems.

We demonstrated the effectiveness of OEL over other methods from the literature on 18 benchmark problems with different dynamics. The following conclusions can be drawn from our experimental studies. Firstly, OEL outperforms TMO approaches on benchmark problems in which the DFFs do not change entirely at random. Secondly, the surrogate model, which is used for estimating a solution's past fitness, plays an important role in OEL as it provides OEL with the training data for fitness time series prediction. Thirdly, the ensemble learning mechanism in OEL reduces the MSE in predicting a solution's future fitness, which leads to better robust solutions than when only one learning model is used for fitness time series prediction. As for the selection of learners in the ensemble in OEL, it is good to have a diverse set of learners in the hope that the prediction errors among different learners are uncorrelated. Finally, the problem parameter time window T when maximizing 'average fitness' or the problem parameter fitness threshold V when maximizing 'survival time' has a huge influence on the performance of OEL.

Currently in OEL, we model a solution's fitness over time as a time series and predict a solution's future fitness based on the ensemble, which is trained on the corresponding fitness time series. For the future work, it would be interesting to directly model the DFFs as a time series and predict future DFFs, which are then used to evaluate a solution's future fitness directly. In addition, we set the weight for each learner in the ensemble to be the same and constant over time. This can be suboptimal when future fitness prediction errors among different learners are correlated or the dynamics of a solution's fitness varies from one region to another in the solution space. Therefore, updating the weights in the ensemble adaptively over time and setting the weights according to a solution's position in the solution space will be an interesting work to pursue. Finally, applying OEL to real world applications is of great interest.

CHAPTER 6

DISTRIBUTION PREDICTION FOR HANDLING ENVIRONMENTAL CHANGES IN ROBUST OPTIMISATION OVER TIME

In Chapter 5, we investigated the method for ROOT problems, and we developed an algorithm framework, i.e., OEL, for solving ROOT problems. An important aspect in solving ROOT problems and also more general DOPs using EAs is handling the environmental changes in DFFs. To be more specific, EAs are adapted via, e.g., initializing a new population, increasing the mutation rate, etc., right after an environmental change happens. In OEL developed in Chapter 5, whenever an environmental change happens in ROOT problems, a new population is randomly initialised in the population-based optimizer, and one individual is replaced with the best solution found in terms of fitness at last time step. The way OEL handles an environmental change is inefficient, and one obvious reason is that most historical evaluation information is simply discarded, which otherwise could be exploited to better prepare the population-based optimizer for the new environment.

In this chapter, we investigate the dynamic handling strategy, i.e., adapting EAs right after an environmental change, for ROOT problems. Particularly, we are interested in adapting EAs by generating a promising initial population whenever an environmental change happens. The rest of this chapter is organised as follows. Section 6.1 briefly reviews existing dynamic handling strategies in EDO for solving various types of DOPs and DMOPs, and the main weaknesses of different types of dynamic handling strategies are identified. In Section 6.2, we develop a novel dynamic handling strategy for solving ROOT problems, in which distributions of good solutions are estimated for previous time steps and predicted for future time steps. Some experiments are conducted to evaluate the effectiveness of our dynamic handling strategy against existing dynamic handling strategies on ROOT problems in Section 6.3. Finally, a summary is given in Section 6.4.

6.1 Existing Strategies in EDO for Handling Environmental Changes in DOPs

A general assumption in EDO is that past evaluation information is helpful for the current optimisation. Otherwise, there is no reason to use other strategies rather than restarting EAs every time the DFF changes. Assuming that learning from the past is indeed helpful, various strategies have been proposed to adapt static EAs to solving DOPs (Nguyen et al. [2012a]). Out of those strategies, a major effort has been devoted to handling the environmental change in DFFs, from the perspective of EAs, immediately after the environmental change is detected or informed (Cobb [1990], Branke [1999], Hu and Eberhart [2002], Hatzakis and Wallace [2006], Rossi et al. [2008], Woldesenbet and Yen [2009], Zhou et al. [2013]).

Strategies for EAs to handle environmental changes in DFFs can be generally divided into two groups, i.e., *implicit prediction strategies* and *explicit prediction strategies*. In *implicit prediction strategies*, many ad hoc heuristics have been proposed to adapt EAs immediately after the environment changes. Typical examples are introducing diversity when changes occur (Cobb [1990], Hu and Eberhart [2002]), memory-based strategies (Branke [1999], Yang and Yao [2008]), and adaptation of population based on historical evaluation information (Woldesenbet and Yen [2009]). More recently, a lot of the strategies have been focusing on explicitly building a time series learning model on previous evaluation information and using the trained model to handle the environmental change. We term those strategies as *explicit prediction strategies*. Typical examples are (Hatzakis and Wallace [2006]) and (Simões and Costa [2009]), with a strategy (Zhou et al.

[2013]) in the DMOPs literature.

The problems with *implicit prediction strategies* are either a relatively strong assumption is made in the DFF or a small amount of previous evaluation information is actually used. More specifically, for strategies about introducing diversity when changes occur (Cobb [1990], Hu and Eberhart [2002]), it is implicitly assumed that a new optimum is somewhere near the population right before the environmental change. Moreover, only evaluation information from the last time step is used. For memory-based strategies, it is assumed that the DFF changes recurrently, and therefore it is useful to introduce some previous good solutions to the population right after environmental changes. For strategies of adaptation of population based on historical evaluation information, only the evaluation information from the latest time steps is actually used (Woldesenbet and Yen [2009], Bui et al. [2012]). Using evaluation information only from the latest one or two time steps can be problematic as this can potentially prevent the decision maker from discovering a long-term dynamic of DFF, e.g., a recurrent dynamic with a period of 10 time steps.

By building an explicit learning model on historical evaluation information, *explicit prediction strategies* can hopefully deal with more types of dynamic environments than *implicit prediction strategies*. Yet, existing *explicit prediction strategies* made their predictions based on the training data consisting of only best solutions found for each previous time step (Hatzakis and Wallace [2006], Simões and Costa [2009], Zhou et al. [2013]). Using only best solutions found previously can be problematic if the best solutions are far from any true optimum, which is very likely to happen if there is not enough computational resource within one time step. More importantly, other evaluation information accumulated in the past is simply discarded, which otherwise can be helpful if an appropriate learning mechanism is developed.

To summarise different strategies for EAs in EDO to handle environmental changes, the main weakness of *implicit prediction strategies* is that an explicit learning model is lacking, and a lot of historical evaluation information is simply discarded. The main weakness of *explicit prediction strategies* is that learning models are built on the training data consisting of only best solutions found in the past, which can be problematic if best solutions found in the past are poor approximations to any true optimum.

Motivated by the main problems discussed in existing prediction strategies (i.e., both *implicit prediction strategies* and *explicit prediction strategies*) for EAs to handle environmental changes in DFFs, we propose a new prediction strategy in this chapter. In contrast to existing prediction strategies, our strategy builds an explicit learning model on distributions of good solutions over time. The proposed prediction strategy is used to reinitialize a population in EAs for ROOT problems whenever the environment changes. By building an explicit learning model on distributions of good solutions over time rather than best solutions found over time, it is hoped that our prediction strategy, compared to existing strategies, can deal with more types of dynamic environments, learn more useful information from historical evaluation information, and provide relatively accurate predictions even when there is little computational resource available for each time step. It should be noted that it is easy to adapt our prediction strategy to other types of DOPs and even DMOPs as we are focusing on the mechanism of learning from historical evaluation information. The details of the proposed prediction strategy for handling environmental changes in ROOT are described in the next section.

6.2 Distribution Prediction in Robust Optimisation Over Time

In this section, the proposed prediction strategy for EAs to handle environmental changes in ROOT is described in detail. The main idea in our strategy is to build a time series model on the distributions of good solutions over time. The predicted distributions made by the trained model are then used to initialise some individuals at the beginning of each time step. Those individuals sampled from the predicted distributions are aimed to be a good starting point for EAs when the DFF changes, especially when there is not enough computational resource at each time step. In contrast to previous related work in which positions of optimal solutions are predicted, we predict the distributions of good solutions, and therefore we term our strategy as Distribution Prediction Strategy (DPS). There are three steps in DPS, namely estimating distributions of good solutions, prediction of distributions of good solutions, and population initialization, which are described in sequence in the following.

6.2.1 Estimating Distributions of Good Solutions

Without loss of generality, suppose at time step t ($1 \leq t \leq N$), the decision maker has evaluated Δe number of solutions, which form the database $\mathcal{D}_t$ ($\mathcal{D}_t = \{(\mathbf{x}, f_t(\mathbf{x}), t)_j\}$, $1 \leq j \leq \Delta e$). We introduce a parameter ρ_s[1], $0 < \rho_s \leq 1$, that controls the proportion of evaluated solutions in database $\mathcal{D}_t$ that is used to estimate a distribution of good solutions at time step t. The distribution of good solutions at time step t is estimated by fitting a multivariate Gaussian distribution $\mathcal{N}(\mathbf{m}_t, \mathbf{C}_t)$ to N_s ($N_s = \Delta e * \rho_s$) solutions selected from $\mathcal{D}_t$, where $\mathbf{m}_t$ is the mean vector, $\mathbf{C}_t$ is the covariance matrix, and N_s best solutions in terms of fitness f_t are selected from $\mathcal{D}_t$. We assume the covariance matrix $\mathbf{C}_t$ to be diagonal in DPS, so that $\mathbf{C}_t$ can be represented as a vector $(\sigma^2_{t1}, \sigma^2_{t2}, ...\sigma^2_{td})$, where σ^2_{ti}, $1 \leq i \leq d$, is the variance for the ith dimension of the distribution at time step t, and d is the dimension of $\mathbf{x}$.

The mean $\mathbf{m}_t$ of the multivariate Gaussian distribution of good solutions at time step t is estimated as the sample mean:

$$\mathbf{m}_t = \frac{1}{N_s}\sum_{j=1}^{N_s}\mathbf{x}_{t(j)}, \tag{6.1}$$

where $\mathbf{x}_{t(j)}$ is the jth best solution in terms of fitness f_t in database $\mathcal{D}_t$.

Since the covariance matrix is assumed to be diagonal, the covariance matrix $\mathbf{C}_t$ can be estimated dimension by dimension. The variance σ^2_{ti} for the ith ($1 \leq i \leq d$) dimension in $\mathbf{C}_t$ is estimated as follows:

$$\sigma^2_{ti} = \frac{1}{N_s - 1}\sum_{j=1}^{N_s}(x_{t(j)i} - m_{ti})^2, \tag{6.2}$$

[1]When maximizing 'average fitness' in ROOT, ρ_s is used. When maximizing 'survival time' in ROOT, the proportion of solutions in database $\mathcal{D}_t$ used for estimation of a distribution of good solutions is set to $max(\rho_s, \rho_v)$, where ρ_v is the proportion of solutions in $\mathcal{D}_t$ with fitness no smaller than the fitness threshold V.

where $x_{t(j)i}$ is the ith dimension of the jth best solution in terms of fitness f_t in database $\mathcal{D}_t$. m_{ti} is the ith dimension of the estimated mean $\mathbf{m}_t$ in Equation 6.1.

We estimate a multivariate Gaussian distribution of good solutions at each previous time step. As a result, at the beginning of time step t, $t \geq 2$, we have a time series of Gaussian distributions ($\mathcal{N}(\mathbf{m}_1, \mathbf{C}_1)$, $\mathcal{N}(\mathbf{m}_2, \mathbf{C}_2)$, ..., $\mathcal{N}(\mathbf{m}_{t-1}, \mathbf{C}_{t-1})$).

6.2.2 Prediction of Distributions of Good Solutions

Given distributions of good solutions for previous time steps, we are interested in the prediction of distributions of good solutions for future time steps. Essentially, this is a time series prediction problem, and any suitable time series prediction technique can be applied given corresponding prior knowledge. We employ the ensemble model used in OEL developed in Chapter 5 for the time series prediction task. The reasons are two-fold. Firstly, the ensemble contains three computationally cheap learners (i.e., AR (Box et al. [1994]), NN (Farmer and Sidorowich [1987]), and MA (Box et al. [1994])), and therefore the computational time to train it is much less than other more complex non-linear models. This property is desirable if not much computational resource is given at a time step. Furthermore, little prior knowledge about the dynamic of distributions of good solutions is available, and an ensemble may provide more accurate predictions compared to a single model.

For a time series of Gaussian distributions, there are two ways to build the corresponding ensemble. We can build a multivariate time series model by treating the mean together with the covariance matrix (the diagonal covariance matrix is reduced to a vector of variances) as a whole vector. Alternatively, for each dimension of the whole vector, we can build a univariate time series model. Therefore, with the dimension of $\mathbf{x}$ being d, we have $2 * d$ independent ensemble models. In our approach, we follow the latter way. The reason is that by treating each dimension independently in the whole vector, as suggested in (Hatzakis and Wallace [2006]), the prediction error in one dimension cannot propagate to others, which can hopefully improve the overall prediction accuracy.

Let $(y_1, y_2, ...y_L)$ be a scalar time series of length L. The prediction task of the ensemble is to predict future values of y: y_{L+i}, $i \geq 1$. The prediction made by the ensemble is simply the average of the prediction made by each learner in the ensemble:

$$\hat{y}_{L+i} = \frac{1}{k}\sum_{j=1}^{k} \hat{y}^{j}_{L+i}, \tag{6.3}$$

where k is the number of learners in the ensemble, which is 3 in this study. $\hat{y}_{L+i}$ is the prediction of the ensemble at time step $L+i$, and $\hat{y}^{j}_{L+i}$ is the prediction of the jth learner in the ensemble at time step $L+i$.

Each learner in the ensemble is trained independently on the whole training data. For more details of the learning models (i.e., AR, NN, and MA) used in the ensemble for predicting a distribution, readers are referred to Appendix .8.

6.2.3 Population Initialization

Assume at the beginning of time step t we have accumulated enough historical evaluation information, i.e., $t > L_{min}$, where L_{min} is the length of time series that is required to train the ensemble model. We first train an ensemble model for each dimension of the combined vector of the Gaussian mean and the Gaussian variances, and the future values of each dimension are predicted. We then use the predicted Gaussian distributions of good solutions to initialise some individuals for the initial population at time step t. It should be noted that the predicted mean or variance can have improper values. Therefore, we need to reset the mean or variance when they get out of range. Without loss of generality, assuming the ith ($1 \leq i \leq d$) dimension of $\mathbf{x}$ is within the search interval $[a, b]$, the following resetting rule applies to the ith dimension $\hat{m}_{(t+j)i}$ of the predicted mean $\hat{\mathbf{m}}_{t+j}$, $j \geq 0$, at time step $t+j$:

$$\hat{m}_{(t+j)i} = \begin{cases} a, & \text{if } \hat{m}_{(t+j)i} < a, \\ b, & \text{if } \hat{m}_{(t+j)i} > b, \\ \hat{m}_{(t+j)i}, & \text{otherwise.} \end{cases} \tag{6.4}$$

In other words, we want to make sure the predicted mean is within the search range for each dimension. Also, the following resetting rule applies to the ith dimension $\hat{\sigma}^2_{(t+j)i}$ of the predicted variances $\hat{\mathbf{C}}_{t+j}$, $j \geq 0$, at time step $t+j$:

$$\hat{\sigma}^2_{(t+j)i} = \begin{cases} \theta_{var} * \frac{b-a}{3}, & \text{if } \hat{\sigma}^2_{(t+j)i} < \theta_{var} * \frac{b-a}{3}, \\ \frac{b-a}{3}, & \text{if } \hat{\sigma}^2_{(t+j)i} > \frac{b-a}{3}, \\ \hat{\sigma}^2_{(t+j)i}, & \text{otherwise,} \end{cases} \tag{6.5}$$

where θ_{var} is a parameter within $(0, 1)$. In our approach, we set θ_{var} to 0.2. The reason to reset the predicted variance (if out of range) is that we want $\hat{\sigma}^2_{(t+j)i}$ to be neither too large nor too small. It is very likely to have the ith dimension of the corresponding sampled individual out of the search interval $[a, b]$ if $\hat{\sigma}^2_{(t+j)i}$ is too large. On the other hand, if $\hat{\sigma}^2_{(t+j)i}$ is too small, the ith dimension for most sampled individuals would be within a very small range compared to the interval $[a, b]$, which might offer no additional exploitation advantage but harm the exploration ability for the initial population at the beginning of time step t.

Assuming the population size is P_{size}, we introduce a parameter ρ_{init}, $0 < \rho_{init} \leq 1$, which controls the proportion of population that is initialised using samples from the predicted distributions of good solutions. Without loss of generality, we assume totally N_{init} ($N_{init} = \lceil P_{size} * \rho_{init} \rceil$) individuals are initialised using samples from the predicted Gaussian distributions for the initial population at the beginning of time step t. The rest $P_{size} - N_{init}$ individuals are uniformly randomly generated across the search space. Since in ROOT the objective in maximizing 'average fitness' is different from that in maximizing 'survival time', we describe the corresponding initialisation procedure respectively in the following.

6.2.3.1 Initialization for Maximizing 'Average Fitness'

The objective in maximizing 'average fitness' at time step t is to find a solution that maximizes the robustness 'average fitness' with the predefined time window T. At the beginning of time step t, we sample some individuals using each predicted Gaussian distribution $\mathcal{N}(\hat{\mathbf{m}}_{t+j}, \hat{\mathbf{C}}_{t+j})$, $0 \leq j \leq T-1$, respectively. Besides, the position of each Gaussian mean is sampled first. It is hoped, in this way, promising areas in terms of 'average fitness' can be located. The initialization procedure of DPS at the beginning of time step t for maximizing 'average fitness' is further summarised in Algorithm 2.

Algorithm 2 Initialisation Procedure in DPS for Maximizing 'Average Fitness' or 'Survival Time ' at the Beginning of Time Step t ($t > L_{min}$).

```
 1: INPUT: The population size $P_{size}$; the parameter $\rho_{init}$, i.e., $N_{init}$ ($N_{init} = \lceil P_{size} * \rho_{init} \rceil$) individuals are initialised using predicted Gaussian distributions; The time window $T$ and the predicted Gaussian distributions of good solutions: $\mathcal{N}(\hat{\mathbf{m}}_{t+j}, \hat{\mathbf{C}}_{t+j})$, $0 \leq j \leq T-1$, or the fitness threshold $V$ and the predicted Gaussian distribution $\mathcal{N}(\hat{\mathbf{m}}_t, \hat{\mathbf{C}}_t)$.
 2: OUTPUT: The initial population $\mathcal{P}_t^{init}$ at time step $t$.
 3: Begin
 4: $\mathcal{P}_t^{init} \leftarrow$ empty set;
 5: for $i = 1 \rightarrow (P_{size} - N_{init})$ do
 6:     Sample $\mathbf{x}$ uniformly randomly across the search space;
 7:     $\mathcal{P}_t^{init} \leftarrow \mathcal{P}_t^{init} \cup \mathbf{x}$ ;
 8: end for
 9: if maximizing 'average fitness' then
10:     $count = 0$;
11:     for $j = 1 \rightarrow T$ do
12:         $\mathcal{P}_t^{init} \leftarrow \mathcal{P}_t^{init} \cup \hat{\mathbf{m}}_{t+j-1}$;
            $count = count + 1$;
13:         if $count == N_{init}$ then
14:             Terminate;
15:         end if
16:     end for
17:     while $count < N_{init}$ do
18:         for $j = 1 \rightarrow T$ do
19:             Sample $\mathbf{x} \sim \mathcal{N}(\hat{\mathbf{m}}_{t+j-1}, \hat{\mathbf{C}}_{t+j-1})$;
20:             $\mathcal{P}_t^{init} \leftarrow \mathcal{P}_t^{init} \cup \mathbf{x}$;
21:             $count = count + 1$;
22:             if $count == N_{init}$ then
23:                 Terminate;
24:             end if
25:         end for
26:     end while
27: end if
28: if maximizing 'survival time' then
29:     $\mathcal{P}_t^{init} \leftarrow \mathcal{P}_t^{init} \cup \hat{\mathbf{m}}_t$;
30:     for $i = 2 \rightarrow N_{init}$ do
31:         Sample $\mathbf{x} \sim \mathcal{N}(\hat{\mathbf{m}}_t, \hat{\mathbf{C}}_t)$;
32:         $\mathcal{P}_t^{init} \leftarrow \mathcal{P}_t^{init} \cup \mathbf{x}$;
33:     end for
34: end if
35: Output the initial population $\mathcal{P}_t^{init}$;
36: End
```

6.2.3.2 Initialization for Maximizing ‘Survival Time’

The objective in maximizing ‘survival time’ at time step t is to find a solution that maintains its fitness above the predefined threshold V as long as possible. In order for the solution to have a large ‘survival time’, its fitness for the current fitness function f_t must be no smaller than V. Therefore, we sample individuals only using the predicted Gaussian distribution for the current time step t: $\mathcal{N}(\hat{\mathbf{m}}_t, \hat{\mathbf{C}}_t)$. Besides, the position of the Gaussian mean $\hat{\mathbf{m}}_t$ is sampled first. The initialization procedure of DPS at the beginning of time step t for maximizing ‘survival time’ is further summarised in Algorithm 2.

6.2.4 DPS for ROOT

The objective in ROOT is to find a robust solution within a computational budget Δe at each time step. We assume that the environmental change is informed to the decision maker. Without loss of generality, we illustrate in Algorithm 3 how DPS is incorporated into a population-based ROOT algorithm at time step t. Note that time step t starts from 1 and increases by 1 every time the DFF changes. DPS is not used until t is greater than L_{min}. When t is not greater than L_{min}, the initial population at each time step is randomly initialised.

Algorithm 3 DPS for ROOT at Time Step t.

```
1: Begin
2: if $t \leq L_{min}$ then
3:     Initialise the population $\mathcal{P}_{init}$ uniformly randomly across the search space;
4: else
5:     Predict future distributions of good solutions using the prediction procedure in DPS; Initialise the population $\mathcal{P}_{init}$ using the initialisation procedure in DPS;
6: end if
7: Search for a robust solution with the initial population being $\mathcal{P}_{init}$; Output the found robust solution after a certain amount of computational resource is exhausted.
8: Estimate the distribution of good solutions for time step $t$: $\mathcal{N}(\mathbf{m}_t, \mathbf{C}_t)$;
9: End
```

6.3 Experimental Studies

The goal of the experimental studies is to show the effectiveness of our DPS strategy compared to existing strategies on handling environmental changes in ROOT problems. Particularly, we are interested in the performance, in terms of the determined solution's robustness at each time step, of these strategies in different dynamic environments under different budgets of computational resource. The budget of computational resource at one time step is measured in the number of fitness evaluations Δe. The reason for such an experimental design is that different dynamics can have a large impact on the learning mechanisms in different strategies. Also, how much computational resource is available at each time step can also influence the way in handling environmental changes. For instance, if there is enough computational budget to determine a solution at each time step, a random restart strategy might be as good as, or even better than, other strategies that try to learn from the past. Also, we would like to demonstrate the effectiveness of DPS by showing that DPS can strike a better balance between diversity and convergence in the initial population than existing strategies.

6.3.1 Experimental Settings

6.3.1.1 Test Problems

There are many benchmarks for studying DOPs in EDO (Branke [1999], Morrison and De Jong [1999], Jin and Sendhoff [2004], Li et al. [2008]). Usually, those benchmarks are constructed by first defining a baseline fitness function and then the dynamics for changing the baseline fitness function. One of the most widely used baseline fitness functions (Branke [1999], Morrison and De Jong [1999], Nguyen et al. [2012a]), takes a cone-like form as the following:

$$f_t(\mathbf{x}) = \max_{i=1}^{i=m}\{h_t^i - w_t^i * ||\mathbf{x} - \mathbf{c}_t^i||_2\}, \tag{6.6}$$

where scalars h_t^i, w_t^i, and vector $\mathbf{c}_t^i, \mathbf{c}_t^i = (c_1^i, ..., c_d^i)$, denote the height, width, and centre of the ith peak function at time step t. $\mathbf{x}$ is a vector of decision variables with d dimensions, and m denotes the total number of peaks. The baseline fitness function in Equation 6.6 is first defined in (Morrison and De Jong [1999]). In order to represent a majority of benchmarks studied in EDO, the baseline fitness function in Equation 6.6 is chosen as the first baseline fitness function in the experimental studies.

By demonstrating that it is difficult, if not impossible, to calculate the absolute best robustness using existing EDO benchmarks, we have developed two baseline fitness functions in Chapter 4 that enable the calculation of the absolute best 'average fitness' and the absolute best 'survival time' respectively. The absolute best robustness (either 'average fitness' or 'survival time') in a benchmark is necessary for evaluating and comparing ROOT methods as it shows the gap between the absolute best performance and the performance of an algorithm. Therefore, the two baseline fitness functions developed in Chapter 4 are employed in the experimental studies. The baseline fitness function for maximizing 'average fitness' is:

$$f_t(\mathbf{x}) = \frac{1}{d}\sum_{j=1}^{d} \max_{i=1}^{i=m}\{h_t^{ij} - w_t^{ij} * |x_j - c_t^{ij}|\}\,, \tag{6.7}$$

and the baseline fitness function for maximizing 'survival time' is:

$$f_t(\mathbf{x}) = \min_{j=1}^{j=d}\{\max_{i=1}^{i=m}\{h_t^{ij} - w_t^{ij} * |x_j - c_t^{ij}|\}\}\,, \tag{6.8}$$

where, in both baseline fitness functions, scalars h_t^{ij}, w_t^{ij}, and c_t^{ij} denote the height, width, and centre of the ith peak function for the jth dimension at time step t. $\mathbf{x}$ is the decision variables with d dimensions ($\mathbf{x} = (x_1, ..., x_d)$), and m is the number of peaks for each dimension.

The 6 different dynamics suggested in the dynamic benchmark generator (Li et al. [2008]), which have been described in Section 4.2.2, are applied to change the baseline fitness functions in Equations 6.6, 6.7, and 6.8.

Applying the 6 different dynamics and the rotation technique to the three baseline fitness functions in Equations 6.6, 6.7, and 6.8, we get 18 different DFFs. We refer each of them as TP_{ij}, $i = 1, 2, 3$ & $j = 1, 2, ..., 6$, where TP_{ij} represents the benchmark which is generated by applying the jth dynamic to the ith baseline fitness function. TP_{1j}, $j = 1, 2, ..., 6$, are used for maximizing both 'average fitness' and 'survival time', while TP_{2j}, $j = 1, 2, ..., 6$, are used only for maximizing ‘average fitness’ and TP_{3j}, $j = 1, 2, ..., 6$, are used only for maximizing ‘survival time’. For each TP_{ij} in this chapter, we generate 200 consecutive fitness functions as follows. For the first fitness function, we randomly initialize the height and width in their corresponding ranges and the centre across the solution space. To generate the next fitness function, we apply the jth dynamic to the current height, width, and rotation angle. The centre of the next fitness function is obtained by rotating the centre of the current fitness function using the updated rotation angle. Whenever the height and width get out of their corresponding ranges, we reset them to their upper limits (if larger than upper limits) or lower limits (if lower than lower limits). Note the exceptions that, for the *chaotic* change, the centre is updated dimension by dimension using Equation 4.11 rather than the rotation technique. For the *recurrent* change and the *recurrent with noise* change, the rotation angle θ is fixed to $\frac{2\pi}{P}$ where P is the period and the centre is rotated following a fixed direction. A summary of all test problem parameters are presented in Table 6.1. It should be noted that the reason to set the time window T to 1 in Table 6.1 is that we would also like to evaluate DPS on TMO problems other than ROOT problems (TMO problems are equivalent to ROOT problems when maximizing ‘average fitness’ with the time window T being 1).

6.3.1.2 Performance Measurement

We use the performance measurement defined in Equation 4.18 in Section 4.3.1.3, and the reason is the same as that explained in Section 4.3.1.3. Given a sequence of fitness functions $(f_1, f_2, ...f_N)$, the performance measurement is rewritten in the following for clarity:

$$Performance^{ROOT} = \frac{1}{N}\sum_{i=1}^{N} F(\mathbf{x}_i), \tag{6.9}$$

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

Note that the performance measurement in Equation 6.9 is dependent on parameter settings, being either T if ‘average fitness’ is considered or V if ‘survival time’ is investigated. Therefore, in order to compare an algorithm’s ROOT ability

Table 6.1: Parameter settings of the test problems

number of peaks, m	5
number of dimensions d	2
search range	$[-25, 25]$
height range	$[30, 70]$
width range	$[1, 13]$
angle range	$[-\pi, \pi]$
$height_{severity}$	5.0
$width_{severity}$	0.5
$angle_{severity}$	1.0
initial angle θ	0
γ	0.04
γ_{max}	0.1
chaotic constant A	3.67
period P	12
$noise_{severity}$	0.8
number of dimensions l for rotation	2
time window T	1,3,5
fitness threshold V	40,45,50

comprehensively, results should be reported under different settings of T and V.

6.3.1.3 ROOT Solver

We use the algorithm framework OEL developed in Chapter 5 as the ROOT solver, which is implemented as follows. Within the framework, the PSO (Clerc and Kennedy [2002]) is used as the population-based search algorithm. The swarm population size is 50 ($P_{size} = 50$). The constants c_1 and c_2, which are used to bias a particle's attraction to the personal best and the global best, are both set to 2.05. The constriction factor χ takes a value of 0.729844. The velocity of particles are constricted within the range $[-v_{max}, v_{max}]$. The value of v_{max} is set to the range of the search space, which is 50 in our case. The global surrogate model in the framework is the RBFN, which is used to approximate a solution's previous fitness. In the RBFN, the K-means (MacQueen et al. [1967]) algorithm is first used to determine the centre of each basis function, where 500 iterations are executed. The number of basis functions is arbitrarily set to 10. A normalised

Gaussian radial basis function is used, and a density estimation technique is used to determine the width for each basis function. Finally, a least square solution is obtained for the weight of each basis function. Moreover, the learning models in the ensemble of the framework for predicting a solution's future fitness are AR, NN, and MA. The length of time series of a solution's fitness over time is set to 20. The embedding size for the time series in AR is set to 4 (i.e., the order of AR is 4). The coefficients of AR are determined using the least square technique. The embedding size for the time series in NN is 4, and the size of neighbourhood in NN is 3. For MA, the average of the latest 3 data points in the time series is used to give the prediction for future data points. For more information about the algorithm framework OEL, readers are referred to Chapter 5.

It should be noted that if we are dealing with 'average fitness' and the time window T is set to 1, i.e., we are essentially solving TMO problems, there is no need to predict a solution's future fitness. In other words, when TMO, the algorithm framework OEL developed in Chapter 5 at a time step is essentially a population-based search algorithm, which is the PSO (Clerc and Kennedy [2002]) in our implementation.

6.3.1.4 Strategies in Comparison

DPS proposed in this chapter is compared with four representative strategies from the literature for handling environmental changes in DFFs, i.e., generating a new population when a change occurs. The first strategy is called Random Restart Strategy (RRS), in which the whole population is reinitialised uniformly randomly across the solution space when a change occurs.

The second strategy is about introducing diversity to the population when a change occurs (Hu and Eberhart [2002]). Basically, when a change occurs, we initialize a new population by randomly copying half of the population right before the change and generating the other half uniformly randomly across the solution space. We name this strategy Last Population Strategy (LPS) as only evaluation information from the last time step is used. Similar strategies can also be found in (Yang and Li [2010], Jin et al. [2012]).

Compared to the second strategy, the third strategy learns more information from historical evaluation information as best solutions found in previous time steps other than only the last time step are introduced to the population when a change occurs. More specifically, when a change occurs, half of the population is randomly reinitialised; 10% of the population is copied from best solutions found for each previous time step starting from the last time step backwards (one best solution for one time step); and the rest 40% is randomly sampled from the population right before the change. If not enough previous best solutions have been stored, the slots will be assigned to individuals randomly copied from the population right before the change. Since we transfer best solutions found at previous time steps into the initial population, we call the third strategy Memory Strategy (MS). Similar memory strategies can also be found in (Branke [1999], Yang et al. [2010]).

The second and the third strategies can be considered as *implicit prediction strategies* as no explicit learning model is employed to learn from the past. The fourth strategy (Hatzakis and Wallace [2006]) investigated belongs to *explicit prediction strategies* as the time series data consisting of best solutions found for each previous time step is used to train an explicit learning model[1] to predict new optimal solutions when the environment changes. The predicted optimal solutions[2] are copied into the initial population for the new time step. Half of the initial population is uniformly randomly generated, and the rest except for the predicted optimal solutions is randomly copied from the population right before the change. Since only the positions of optimal solutions are predicted, we name this strategy Optimum Prediction Strategy (OPS). The strategy in (Zhou et al. [2013]) also predicts the positions of optimal solutions, i.e., the Pareto front solutions, except that it is used for DMOPs.

As for the settings of DPS, we use the estimated Gaussian means and variances from the latest 20 time steps to train the ensemble model, i.e., $L_{min} = 20$. The settings of the ensemble is the same as that in the ROOT solver (i.e., the algorithm framework OEL developed in Chapter 5), described previously in this section. ρ_s and ρ_{init} in DPS take the values of 0.2 and 0.5 respectively. For all the strategies

[1]The same AR model is used as the AR model in the ROOT solver.

[2]When maximizing 'average fitness', T, i.e., the time window size, number of future optimal solutions are predicted (here, we assume that T is smaller than half of the population size P_{size}.), while only one future optimal solution is predicted when maximizing 'survival time'.

in comparison, once the DFF changes, the personal best and the global best in memories of particles in the population are erased, and the velocity of all particles are reinitialised randomly in the interval $[-v_{max}, v_{max}]$.

6.3.2 Simulation Results

All the results reported below are from time step 21 to 100 as the length of time series of distributions of good solutions in DPS is set to 20 (i.e., $L_{min} = 20$). 30 independent runs are conducted. Also, the maximal 'survival time' of a solution is set to 10 as some solution's 'survival time' can be infinity in certain benchmarks. Results with a '+' or '-' in Tables 6.2 and 6.3 are significantly better or worse than those of DPS at the 0.05 significance level using the Wilcoxon rank sum test.

6.3.2.1 Results for Maximizing 'Average Fitness'

The results for maximizing 'average fitness' using different dynamic handling strategies are presented in Table 6.2. We can see that DPS is generally significantly better than RRS under different dynamics when there is not enough computational time to converge to optimal solutions at each time step (i.e., Δe is small). This demonstrates that an initial population produced by DPS at the beginning of a time step is better than a randomly initialised one. When Δe is large, there is enough time for a randomly initialised population to converge to a good solution. In other words, an initial population produced by DPS offers no advantage over a randomly generated one in terms of converging to a good solution (DPS performs comparably to RRS with a worse performance on TP_{25}). We also note that when a DFF changes entirely at random (i.e., for the test problems TP_{13} and TP_{23}), DPS performs slightly worse than RRS as the learning in DPS is simply fitting noise in such a DFF.

Compared to other strategies (i.e., LPS, MS, and OPS) which learn from the past, the performance of DPS is significantly better in many cases under different settings of Δe and different dynamics. This is because DPS strikes a better balance between the diversity and the convergence of the initial population. The better balance of DPS will be demonstrated later in this chapter. The reason why a better balance between the diversity and the convergence of an initial population leads to better performance is as follows. For any dynamic of DFF, when Δe is small, an initial population with a better performance has a higher chance in converging to a good solution. In other words, the convergence of an initial population dominates the performance of the final solution determined at a time step. On the other hand, when Δe is large, the convergence of an initial population may prevent the population from exploring other promising regions. In other words, the diversity of an initial population plays a more important role when Δe is large since the search algorithm has enough time to converge to a good solution at a time step.

The reason why DPS strikes a better balance between the diversity and the convergence of an initial population is due to the following aspects. Firstly, DPS employs an explicit learning model so that the prediction in DPS can be relatively accurate for a range of dynamics, while implicit learning models, e.g., in LPS and MS, may only be effective for certain dynamics of DFF. The second aspect is that DPS builds a learning model on time series of distributions of good solutions, in comparison to time series of best solutions found over time. Time series models built on distributions of good solutions may be more accurate or stable than on best solutions found previously as best solutions found previously can be bad approximations of true optimal solutions especially when Δe is small. The third aspect may be due to the fact that DPS initialises a population that is partly randomly generated and partly generated according to the predicted distributions of good solutions, while in other strategies the initial population is partly generated by copying some individuals from the population right before the environmental change. In cases where the environmental change is so severe that good solutions for the last environment perform badly for the current environment, inheriting some good solutions from the last environment might offer little help in finding good solutions for the current environment.

We also note that in some cases the performance of LPS, MS, or OPS is comparable to or even better than DPS. This happens mostly when $T = 3$ or $T = 5$. This may due to the fact that DPS predicts distributions of good solutions in terms of fitness at each time step rather than a solution's actual 'average fitness'. Currently, DPS tries to initialize good solutions in terms of 'average fitness' by sampling some individuals using each predicted distribution of good solutions in terms of fitness, and this may not be a good approximation in some cases.

6.3.2.2 Results for Maximizing 'Survival Time'

The results for maximizing 'survival time' using different dynamic handling strategies are presented in Table 6.3. DPS significantly outperforms other strategies mostly in cases when Δe is small. The better performance of DPS is primarily due to the better convergence of an initial population produced by DPS as the convergence plays a more important role than diversity when Δe is small.

In other cases, the difference between the performance of DPS and those of other strategies is insignificant. The reasons for this may be the following. Firstly, when maximizing 'survival time', DPS tries to predict a distribution of solutions with fitness no smaller than the fitness threshold V for the current environment. As most times V is a relatively high threshold, a search algorithm may have found only a few solutions with fitness no smaller than V at each previous time step. Therefore, the data used to estimate a distribution of solutions with fitness no smaller than V at each previous time step is sparse, and thus the estimated distribution is inaccurate. As a result, the prediction of DPS is inaccurate. Secondly, DPS predicts a distribution of solutions with fitness no smaller than the fitness threshold V, which is not exactly a distribution of solutions with good 'survival time'. In other words, the predicted distribution given by DPS may not be strongly correlated with the distribution of solutions with good 'survival time'. Finally, even though in some cases DPS gives a relatively accurate prediction and the predicted distribution is strongly correlated with the distribution of solutions with good 'survival time', this might be offset by the relatively unsatisfactory performance, demonstrated in Chapter 5, of the ROOT solver itself in maximizing 'survival time'.

6.3.2.3 Balance of Diversity and Convergence

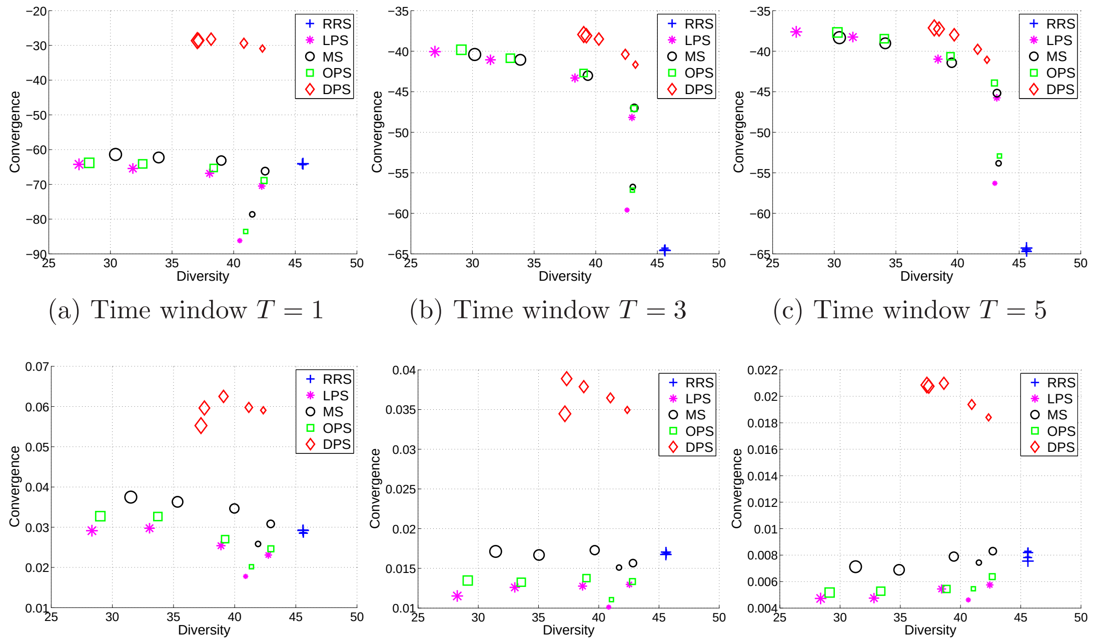
Assuming an initial population $\mathcal{P}^{init}$ of size P_{size} is generated by one of the dynamic handling strategies (i.e., RRS, LPS, MS, OPS, or DPS), we use the Solow-

Table 6.2: The averaged performance measure using Equation 6.9 of different strategies over 30 runs when maximizing 'average fitness' with different settings of the time window T and the computational budget Δe.

Test Problem	Δe	RRS			LPS			MS			OPS			DPS		
		T=1	T=3	T=5	T=1	T=3	T=5	T=1	T=3	T=5	T=1	T=3	T=5	T=1	T=3	T=5
	200	65.26	46.77-	43.94-	65.07-	46.81-	44.01-	65.25	46.87-	44.04-	65.16-	46.96-	44.06-	65.29	47.17	44.30
	400	66.17-	47.44-	44.45	66.16-	47.44-	44.46	66.19	47.47	44.48	66.17-	47.54	44.48	66.23	47.68	44.45
TP_{11}	800	66.86	47.40	44.49	66.86	47.06-	44.36-	66.92	47.30-	44.31-	66.87	47.21-	44.42	66.88	47.57	44.63
	1600	67.17	46.67-	44.06	67.12	46.53-	43.96-	67.18	46.37-	43.96-	67.14	46.48-	44.19	67.14	47.10	44.27
	3200	67.20	46.38	43.93	67.16	46.33-	43.61-	67.20	46.24-	43.64-	67.17	46.24-	43.91	67.19	46.60	44.01
	200	59.15-	50.65-	48.91-	59.05-	50.71-	48.96-	59.19-	50.82-	48.93-	59.03-	50.70-	48.98	59.36	50.96	49.08
	400	60.51	51.16-	49.24	60.45-	51.25	49.19	60.51	51.28	49.19-	60.47-	51.36	49.26	60.54	51.30	49.29
TP_{21}	800	61.40	51.52	49.42	61.37-	51.49	49.35-	61.41	51.49	49.36-	61.39-	51.53	49.38-	61.45	51.53	49.47
	1600	61.88	51.50	49.44	61.86-	51.56	49.47+	61.84-	51.54	49.42	61.83-	51.52	49.35	61.89	51.54	49.39
	3200	61.97	51.52-	49.33	61.93-	51.47-	49.31	61.97	51.49-	49.24-	61.95-	51.47-	49.27	61.99	51.61	49.34
	200	51.05-	21.51-	17.68-	51.00-	21.56-	17.45-	52.20-	22.38-	17.99	51.17-	22.07-	17.57-	53.56	23.04	18.21
	400	57.61-	24.19	19.32	57.95-	24.11	19.10	58.02-	24.50	19.23	57.85-	24.21	19.21	58.32	24.37	19.34
TP_{12}	800	61.79-	25.49	19.88	61.85-	25.52	19.67	61.93	25.61	19.96	61.81-	25.48	19.89	62.01	25.52	19.98
	1600	63.62-	25.86	20.19	63.65-	25.73	20.09	63.67	25.65	20.24	63.65	25.96	20.17	63.74	25.77	20.32
	3200	63.97	25.89+	20.15	64.05	25.77	20.05	64.04	25.43	20.11	64.00	25.78	20.12	64.04	25.60	20.05
	200	59.20-	36.87-	33.42	59.02-	36.57-	33.56	59.17-	36.84-	33.23-	58.97-	36.83-	33.55	59.39	37.21	33.76
	400	61.97	37.58-	34.12	61.84-	37.82	34.20	61.94-	37.82	33.98	61.97	37.81	34.03	62.05	37.90	34.26
TP_{22}	800	63.95	38.23-	34.15	63.89	38.39	34.27	63.94	38.66	34.19	63.90	38.26	34.20	63.94	38.57	34.20
	1600	64.86	38.51	33.91	64.82	38.48	33.93	64.82	38.35	33.96	64.88	38.57	33.90	64.85	38.51	33.80
	3200	65.11	38.27	33.92	65.12	38.34	33.71	65.13	38.50	33.79	65.11	38.40	33.98	65.13	38.47	33.84
	200	58.82	30.08+	24.56	58.22-	29.74	24.43	58.55	30.03	24.53	58.36-	29.67	24.58	58.62	29.80	24.52
	400	60.81+	30.57	24.95	60.44	30.34	24.83	60.52	30.34	24.94	60.40	30.47	24.81	60.55	30.45	24.76
TP_{13}	800	62.24+	31.09	24.90	62.01	30.86	24.81	62.11	31.10	25.00+	61.96	30.83	24.89	61.98	31.08	24.67
	1600	62.85	31.24	24.46	62.77	31.07-	24.53	62.84	31.17	24.61	62.61	31.22	24.53	62.70	31.39	24.48
	3200	63.10+	31.38	24.24	62.69	31.08	24.25	62.96	31.15	24.45	62.89	30.95-	24.48	62.83	31.27	24.29
	200	59.77	43.89	40.65	59.58-	43.67	40.54	59.75	43.80	40.58	59.57-	43.65	40.75	59.79	43.78	40.70
	400	61.21	44.20	40.91	61.06-	44.06-	40.65-	61.20	44.11	40.86	61.09-	44.02-	40.84	61.21	44.28	40.93
TP_{23}	800	62.18	44.48	41.02	62.10-	44.44	41.01	62.16	44.37	40.99	62.12	44.37	40.94	62.19	44.46	41.00
	1600	62.65	44.36	40.93	62.57	44.34	40.95	62.59	44.36	40.89	62.57	44.33	40.86	62.63	44.37	40.93
	3200	62.74	44.51	40.94	62.74-	44.37	40.86	62.70-	44.39	40.87	62.68-	44.43	40.98	62.79	44.46	41.00
	200	52.92-	7.09-	-0.07-	51.94-	7.30-	0.25-	52.98-	7.51-	0.63	52.32-	7.93-	0.70	55.28	8.79	1.11
	400	59.33-	10.77-	1.84-	59.19-	11.06-	2.20	59.32-	10.79-	2.25	59.13-	11.09-	2.58	59.92	11.64	2.55
TP_{14}	800	62.88-	12.71	3.61	62.79-	12.41	2.82-	62.92-	12.59	3.02	62.79-	12.46-	2.99	63.09	12.94	3.50
	1600	64.29-	13.06	3.67	64.27-	12.75	3.10-	64.30-	12.90	3.31	64.35	12.83	2.94-	64.42	12.71	3.71
	3200	64.67-	12.85	3.40	64.66-	12.81	2.94	64.71	12.64	3.28	64.65-	12.87	3.26	64.79	13.05	3.50
	200	58.96-	41.45	37.86	58.75-	41.43	38.77+	59.05-	41.95	38.44	58.87-	41.99	38.50	59.40	41.71	38.10
	400	61.32-	43.05-	39.90-	61.41	44.00+	41.10+	61.40	44.03+	41.16+	61.46	44.17+	40.73	61.46	43.62	40.57
TP_{24}	800	63.20	44.86-	42.34	63.17	44.99	42.70	63.22	45.09	42.42	63.14	45.24	42.63	63.19	45.16	42.61
	1600	64.25	45.13	42.00	64.20-	45.06	42.24	64.23	45.10	41.99	64.17-	45.06	42.22	64.23	45.11	42.16
	3200	64.56	44.86-	41.84	64.50-	44.96	41.92	64.53	45.05	41.95	64.51-	44.97-	42.03	64.56	45.31	42.20
	200	61.00-	31.09-	16.21-	61.14-	30.61-	15.45-	61.08-	30.83-	15.41-	61.27-	31.22-	16.03-	61.83	32.74	17.83
	400	63.31-	33.02-	17.03-	63.49-	33.68-	17.47-	63.54-	33.72-	17.18-	63.57	33.74	17.96	63.67	34.12	18.41
TP_{15}	800	64.83-	34.62-	17.04-	65.06	34.95-	17.79	64.97-	35.16	17.49-	65.06	34.98-	17.74	65.10	35.40	18.23
	1600	65.59-	35.08	16.96-	65.72-	35.43	17.22-	65.69-	35.45	17.35	65.70-	35.59	17.18-	65.84	35.44	17.71
	3200	65.82-	35.02-	17.42	65.93-	35.55	17.70	65.91-	35.64	17.57	65.86-	35.50	17.64	66.02	35.45	17.77
	200	59.22-	43.10	35.19	59.16-	43.06-	35.09	59.29	42.92-	34.81	59.20-	42.87-	34.86	59.37	43.30	35.06
	400	60.67	43.85	36.17	60.67-	43.98	36.47	60.70	43.85	36.34	60.67	43.74	36.64+	60.73	43.87	36.22
TP_{25}	800	61.63+	44.98	38.01	61.59	45.06	38.81+	61.62+	44.89	38.57	61.51	45.10	38.58	61.54	44.88	38.23
	1600	62.15+	45.97+	39.52	62.05	45.85	39.77	62.11	45.93	39.90	62.05	45.90	39.49	62.08	45.68	39.50
	3200	62.29	46.40+	40.22+	62.19-	46.13	39.78	62.28	46.05	39.90	62.22	46.30+	39.81	62.27	45.93	39.65
	200	57.02-	48.29-	44.20-	57.01-	48.24-	44.61-	57.28-	48.77-	45.02-	57.26-	49.05-	45.80	57.94	49.78	46.02
	400	59.14-	49.91-	45.23-	59.24-	50.03-	44.98-	59.30-	50.07-	45.24-	59.25-	50.11-	45.79	59.46	50.45	45.69
TP_{16}	800	60.48-	50.94	46.10-	60.56-	51.27+	46.54	60.56-	51.18+	46.52	60.56-	51.19+	46.44	60.63	50.94	46.49
	1600	61.13	51.92	47.70	61.12-	51.98	48.71+	61.08-	51.87	48.01	61.09-	51.89	48.55+	61.17	51.87	47.92
	3200	61.24	51.86	47.94	61.21-	51.84	48.09	61.20-	51.83	47.99	61.20-	51.73	48.19	61.27	51.87	47.72
	200	63.19-	51.63	47.33-	63.20-	51.54-	47.27-	63.21-	51.61	47.47-	63.24-	51.68	47.60-	63.53	51.72	47.92
	400	64.49-	51.97	47.78-	64.45-	51.92	47.66-	64.44-	51.94	47.80-	64.49-	51.92	48.03	64.56	52.02	48.07
TP_{26}	800	65.33	52.12	47.87-	65.26-	52.05	48.02-	65.29-	52.10	47.94-	65.27-	52.15	48.02-	65.32	52.15	48.24
	1600	65.72	52.25	47.97	65.67-	52.14-	47.71-	65.68-	52.30	47.76	65.68-	52.21	47.91	65.73	52.33	48.02
	3200	65.84	52.27	47.95	65.80-	52.21-	47.74	65.83	52.11-	47.81	65.79-	52.31	48.02	65.86	52.39	47.90

Table 6.3: The averaged performance measure using Equation 6.9 of different dynamic handling strategies over 30 runs when maximizing 'survival time' with different settings of the fitness threshold V and the computational budget Δe.

Test Problem	Δe	RRS			LPS			MS			OPS			DPS		
		V=40	V=45	V=50	V=40	V=45	V=50	V=40	V=45	V=50	V=40	V=45	V=50	V=40	V=45	V=50
TP_{11}	200	2.79	2.13	1.61	2.84	2.15	1.58	2.74-	2.13	1.55-	2.77	2.10-	1.59	2.88	2.20	1.61
	400	2.88-	2.20-	1.61	2.94	2.24	1.59	2.92	2.28	1.58	2.97	2.22-	1.62	2.99	2.29	1.61
	800	3.03	2.19-	1.62	3.02	2.21	1.66+	2.98-	2.22	1.62	2.95-	2.21	1.58	3.08	2.28	1.58
	1600	2.88-	2.16	1.62	2.94	2.18	1.57	2.96	2.17	1.57-	2.92	2.17	1.63	3.00	2.21	1.62
	3200	2.85	2.16	1.61	2.86	2.14	1.57	2.80-	2.15	1.57	2.75-	2.16	1.58	2.91	2.20	1.62
TP_{21}	200	2.82	2.03	1.12-	2.83	2.01	1.12-	2.78	2.04	1.10-	2.80	2.09	1.12-	2.80	2.09	1.18
	400	2.83	2.10	1.23-	2.90	2.12	1.22-	2.87	2.18	1.23	2.89	2.16	1.20-	2.86	2.15	1.27
	800	2.84	2.27	1.27	2.89	2.25	1.27	2.84	2.24	1.26	2.85	2.28	1.28	2.79	2.27	1.28
	1600	2.76	2.30	1.27	2.71	2.31	1.28	2.83+	2.33	1.26	2.80+	2.34	1.26	2.73	2.30	1.27
	3200	2.73	2.32	1.23	2.69	2.23	1.23	2.72	2.26	1.23	2.77	2.28	1.23	2.72	2.27	1.24
TP_{12}	200	1.02-	0.82-	0.62-	1.03-	0.82-	0.61-	1.08-	0.87-	0.65-	1.02-	0.83-	0.60-	1.13	0.93	0.70
	400	1.18-	1.05-	0.84-	1.19-	1.06	0.84-	1.22	1.09	0.86	1.21	1.06	0.85	1.22	1.09	0.87
	800	1.25-	1.15	0.98-	1.27	1.16	0.98-	1.28	1.19	0.99	1.27	1.18	1.00	1.28	1.17	1.00
	1600	1.28	1.19-	1.05	1.29	1.20	1.05	1.29	1.21	1.05	1.29	1.20	1.06	1.30	1.21	1.05
	3200	1.30	1.20	1.06-	1.31	1.21	1.07	1.32	1.23	1.07	1.31	1.20-	1.07	1.31	1.22	1.08
TP_{22}	200	1.08	0.94	0.80	1.08	0.93	0.77-	1.08	0.92-	0.80	1.11	0.92	0.76-	1.09	0.95	0.79
	400	1.12	1.00	0.90	1.11	1.01	0.89	1.12	1.00	0.90	1.12	1.00	0.89	1.12	1.00	0.90
	800	1.12	1.02	0.93	1.12	1.02	0.93	1.11	1.03	0.93	1.12	1.02	0.93	1.11	1.02	0.93
	1600	1.12	1.01	0.95	1.13	1.02	0.95	1.12	1.02	0.95	1.14	1.01	0.95	1.12	1.01	0.95
	3200	1.12	1.01	0.95	1.13	1.02	0.95	1.12	1.02	0.94	1.12	1.02	0.95	1.12	1.01	0.95
TP_{13}	200	1.20	1.06	0.91	1.18	1.02-	0.90	1.21	1.05	0.91	1.21	1.04	0.90	1.18	1.04	0.91
	400	1.23	1.09	1.00	1.23	1.09	0.98	1.22	1.09	0.98	1.23	1.08	0.97-	1.23	1.09	0.99
	800	1.23	1.13	1.02	1.22	1.12	1.00-	1.23	1.11-	1.00	1.22	1.10	1.01	1.23	1.12	1.01
	1600	1.23	1.12	1.02	1.23	1.11	1.01	1.21-	1.12	1.02	1.22	1.12	1.01	1.24	1.12	1.02
	3200	1.24	1.12	1.02	1.21	1.11-	1.01	1.24	1.11-	1.02	1.22	1.12-	1.01	1.22	1.14	1.02
TP_{23}	200	1.31	1.15	0.92	1.30	1.15	0.91	1.30	1.16	0.92	1.29	1.15	0.91	1.30	1.16	0.91
	400	1.32	1.17	0.97	1.32	1.15	0.97	1.31	1.16	0.96	1.31	1.17	0.97	1.31	1.17	0.97
	800	1.31	1.18	0.98	1.32	1.17	0.99	1.30	1.17	0.99	1.33	1.16	0.98	1.32	1.17	0.99
	1600	1.34	1.18	0.99	1.32	1.16-	0.99	1.32	1.16-	1.00	1.32	1.18	0.98	1.33	1.19	0.98
	3200	1.33	1.18	1.00	1.34	1.17	0.98-	1.35	1.16-	1.00	1.33	1.18	0.99	1.32	1.18	1.00
TP_{14}	200	0.98-	0.90-	0.72-	0.96-	0.86-	0.65-	0.98-	0.90-	0.73-	0.97-	0.87-	0.67-	1.02	0.97	0.85
	400	1.02-	1.00-	0.99-	1.03-	1.00	0.98-	1.03	1.00	0.99	1.03	1.00-	0.98-	1.04	1.01	0.99
	800	1.03	1.01	1.00	1.05	1.00	1.00-	1.04	1.01	1.00	1.04	1.00	1.00	1.04	1.01	1.00
	1600	1.02-	1.00	1.00	1.04	1.01	1.00	1.04	1.01	1.00-	1.04	1.01	1.00	1.04	1.01	1.00
	3200	1.03	1.01	1.00	1.03	1.01	1.00	1.04	1.01	1.00	1.04	1.01	1.00-	1.04	1.01	1.00
TP_{24}	200	1.23-	1.14	0.99-	1.27	1.12	0.99-	1.26	1.12	1.00	1.25	1.13	0.99-	1.27	1.14	1.01
	400	1.30-	1.14-	1.06	1.34	1.14-	1.06	1.36	1.18+	1.08+	1.34	1.16	1.06	1.33	1.16	1.07
	800	1.36	1.16-	1.08	1.38	1.19	1.09	1.39	1.21+	1.09	1.40	1.19	1.08	1.37	1.18	1.09
	1600	1.37	1.18	1.09	1.41	1.20	1.09	1.42	1.22	1.10	1.42	1.20	1.08	1.39	1.20	1.09
	3200	1.37-	1.18	1.09	1.42	1.20	1.10	1.40-	1.19	1.11+	1.40-	1.21	1.09	1.44	1.20	1.09
TP_{15}	200	2.32	2.03	1.56	2.31	2.04	1.57	2.32	2.04	1.51-	2.31	2.02	1.55	2.35	2.06	1.56
	400	2.31	2.07	1.59	2.31	2.06	1.62	2.32	2.05	1.60	2.31	2.07	1.60	2.32	2.06	1.60
	800	2.30	2.12+	1.64	2.30-	2.06	1.64	2.31	2.09+	1.64	2.30	2.05	1.64	2.32	2.04	1.63
	1600	2.32	2.08	1.63	2.32	2.06	1.63	2.30	2.06	1.60	2.31	2.06	1.60	2.32	2.04	1.62
	3200	2.27	2.03	1.58	2.29	2.07	1.58	2.27	2.03	1.55-	2.27	2.04	1.58	2.27	2.04	1.60
TP_{25}	200	1.31	0.98	0.86-	1.32	0.99	0.88	1.31	0.99	0.87	1.32	0.97	0.87-	1.30	0.98	0.88
	400	1.34	0.99	0.92	1.34	0.99	0.92	1.36	0.98	0.92	1.36	0.98-	0.92	1.36	0.99	0.93
	800	1.33	0.96	0.93	1.35	0.97	0.92	1.36	0.97	0.92	1.35	0.97	0.93	1.33	0.96	0.93
	1600	1.36	0.96	0.92	1.34	0.96	0.92	1.37	0.98+	0.93	1.34	0.96	0.93	1.35	0.95	0.92
	3200	1.37	0.96	0.92	1.36	0.99	0.92	1.38	1.01+	0.93	1.38	0.99	0.92	1.35	0.98	0.93
TP_{16}	200	2.67	2.16-	1.60-	2.66	2.14-	1.62-	2.66-	2.20-	1.68-	2.68	2.25	1.69-	2.75	2.31	1.80
	400	2.81-	2.30-	1.85-	2.81-	2.35	1.92-	2.83-	2.31	1.93	2.84-	2.33	1.93	2.96	2.36	1.95
	800	3.08	2.41	1.94	3.09	2.45	1.98	3.07	2.47+	1.98	3.04-	2.42	1.95	3.11	2.41	1.97
	1600	3.10	2.54-	1.92	3.13	2.61	1.93	3.11	2.58	1.93	3.15	2.59	1.92	3.09	2.62	1.94
	3200	3.22	2.67-	1.93	3.20	2.74	1.93	3.19	2.74	1.92	3.23	2.74	1.92	3.24	2.74	1.94
TP_{26}	200	2.53	1.99+	1.40	2.51	1.95	1.38-	2.57	1.95	1.36-	2.56	1.93	1.40	2.54	1.94	1.41
	400	2.59	1.98	1.46	2.55	1.96	1.46	2.56	1.98	1.44	2.60	1.95	1.45	2.62	1.99	1.46
	800	2.62	2.05	1.50	2.67	2.03	1.51	2.60	2.01	1.50	2.61	2.05	1.50	2.61	2.06	1.51
	1600	2.60	2.03-	1.55	2.59	2.09	1.55	2.63	2.08	1.55	2.58	2.07	1.54	2.65	2.12	1.53
	3200	2.63	2.06	1.52	2.62	2.03	1.56+	2.59	2.07	1.54	2.62	2.01	1.54	2.59	2.07	1.52



(a) Time window $T = 1$ (b) Time window $T = 3$ (c) Time window $T = 5$

(d) Fitness threshold $V = 40$ (e) Fitness threshold $V = 45$ (f) Fitness threshold $V = 50$

Figure 6.1: The averaged diversity and convergence of the initial population produced by each dynamic handling strategy at the beginning of a time step on TP_{14}. Each data point in the figure is the average over 30 runs and over time step 21 to 100. For each dynamic handling strategy, we plot a data point for every setting of Δe with bigger size for larger Δe.

Polasky measure (Solow and Polasky [1994]) to quantify the diversity of $\mathcal{P}^{init}$. The Solow-Polasky measure outputs a real number in the interval $[1, P_{size}]$, which can be interpreted as the number of different species in the population. The Solow-Polasky measure is defined as:

$$Div(\mathcal{P}^{init}) = \mathbf{e} * \mathbf{M}^{-1} * \mathbf{e}^T, \tag{6.10}$$

where $\mathbf{e}$ is a row vector $(1, 1, ..., 1)$ of length P_{size}, and $\mathbf{e}^T$ is the transpose of $\mathbf{e}$. $\mathbf{M}^{-1}$ is the inverse[1] of the $P_{size} * P_{size}$ square matrix $\mathbf{M}$. The element m_{ij} at the ith row and the jth column of $\mathbf{M}$ is:

$$m_{ij} = exp(-\theta * d(\mathcal{P}^{init}[i], \mathcal{P}^{init}[j])), \forall 1 \leq i, j \leq P_{size}, \tag{6.11}$$

[1]In case $\mathbf{M}$ is singular, the Moore-Penrose pseudoinverse is used.

where $d(\mathcal{P}^{init}[i], \mathcal{P}^{init}[j])$ is a distance measure between the ith individual and the jth individual in $\mathcal{P}^{init}$. In DPS, the distance measure is the Euclidean distance. θ normalises the relationship between the distance measure and the number of different species and is set to 1.0.

The convergence of $\mathcal{P}^{init}$ is defined as the average performance of individuals in $\mathcal{P}^{init}$:

$$Con(\mathcal{P}^{init}) = \frac{1}{P_{size}} \sum_{i=1}^{P_{size}} F(\mathcal{P}^{init}[i]), \tag{6.12}$$

where $F(\mathcal{P}^{init}[i])$ denotes the 'average fitness' or the 'survival time' of the ith individual $\mathcal{P}^{init}[i]$ in population $\mathcal{P}^{init}$.

We present the diversity and convergence results on TP_{14} in Figure 6.1. From Figure 6.1, we can see that DPS generally strikes a better blance between the diversity and the convergence for maximizing 'average fitness' or 'survival time'. To be more specific, when Δe is small, DPS maintains a high convergence without too much loss of diversity, compared to other strategies. As discussed before, high convergence is beneficial for a population to find a good solution within a small computational time. On the other hand, when Δe is large, DPS maintains not only a high convergence but also a high diversity compared to other strategies that learn from the last (i.e., LPS, MS, and OPS). Also, the convergence produced by DPS is much better than that produced by RRS without too much loss in the diversity. Similar results with regard to the diversity and convergence of the initial population produced by different strategies can be observed on other test problems.

6.3.3 More Discussions on DPS

In this subsection, we investigate the influence of the time series prediction model in DPS and the settings of some important parameters in DPS.

6.3.3.1 Influence of the Time Series Prediction Model in DPS

The results of maximizing 'average fitness' and 'survival time' using DPS with different time series prediction models are presented in Tables 6.4 and 6.5 respectively. Three other types of time series prediction models are investigated:

Table 6.4: The averaged performance measure using Equation 6.9 obtained by DPS with different time series prediction models over 30 runs when maximizing 'average fitness' with different settings of the time window T and the computational budget Δe.

Test Problem	Δe	DPS(AR)			DPS(MA)			DPS(NN)			DPS(Ensemble)		
		T=1	T=3	T=5	T=1	T=3	T=5	T=1	T=3	T=5	T=1	T=3	T=5
TP_{14}	200	55.55+	8.19	0.42	53.60-	8.48	0.49	55.86+	8.39	0.83	55.28	8.79	1.11
	400	59.92	11.50	2.63	59.53-	10.99-	2.52	60.22+	11.06-	2.31	59.92	11.64	2.55
	800	63.05	12.83	3.27	62.82-	12.17-	3.08	63.06	12.75	3.33	63.09	12.94	3.50
	1600	64.41	12.88	3.58	64.30-	12.60	3.12-	64.44	13.24+	3.27	64.42	12.71	3.71
	3200	64.71	12.91	3.54	64.67-	12.74	3.27	64.72	12.94	3.56	64.79	13.05	3.50

Table 6.5: The averaged performance measure using Equation 6.9 obtained by DPS with different time series prediction models over 30 runs when maximizing 'survival time' with different settings of the fitness threshold V and the computational budget Δe.

Test Problem	Δe	DPS(AR)			DPS(MA)			DPS(NN)			DPS(Ensemble)		
		V=40	V=45	V=50	V=40	V=45	V=50	V=40	V=45	V=50	V=40	V=45	V=50
TP_{14}	200	1.01	0.97	0.85	0.99-	0.92-	0.74-	1.01-	0.98+	0.88+	1.02	0.97	0.85
	400	1.03	1.00	0.99	1.03	1.01	0.98-	1.03	1.01	1.00+	1.04	1.01	0.99
	800	1.05	1.01	1.00	1.04	1.00	1.00	1.04	1.01	1.00	1.04	1.01	1.00
	1600	1.04	1.01	1.00	1.04	1.01	1.00-	1.04	1.01	1.00	1.04	1.01	1.00
	3200	1.04	1.01	1.00	1.04	1.01	1.00	1.04	1.01	1.00	1.04	1.01	1.00

AR, NN, and MA, the results of which are denoted as DPS(AR), DPS(NN), and DPS(MA) respectively. The original DPS is denoted as DPS(Ensemble) for clarity. Results with a '+' or '-' in Tables 6.4 and 6.5 are significantly better or worse than those of DPS(Ensemble) at the 0.05 significance level using the Wilcoxon rank sum test. From Tables 6.4 and 6.5, we can see that DPS(Ensemble) performs better than DPS(MA) and slightly worse than DPS(NN) (This is because the NN model is more appropriate for a chaotic dynamic than AR and MA in the ensemble). For other test problems, in general, the time series prediction models have some impact on the performance of DPS, and DPS(Ensemble) achieves the best performance most times.

6.3.3.2 Influence of the Parameter ρ_s

We fix the setting of ρ_{init} ($\rho_{init} = 0.5$) and vary the setting of ρ_s ($\rho_s \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$) in DPS. From Figure 6.2, we can see that a smaller value of ρ_s

gives a better performance when Δe is small ($\Delta e = 200$). When Δe is getting larger, the impact of ρ_s on the performance of DPS becomes negligible. The reason is that when Δe is small all evaluated solutions within a time step tend to spread uniformly across the solution space, and a smaller value of ρ_s is therefore better in terms of estimating a distribution of good solutions. On the other hand, when Δe is large, evaluated solutions are biased to promising areas, and therefore different settings of ρ_s result in more or less the same estimated distribution of good solutions. Similar phenomenon can be observed on other test problems when maximizing either 'average fitness' or 'survival time'.

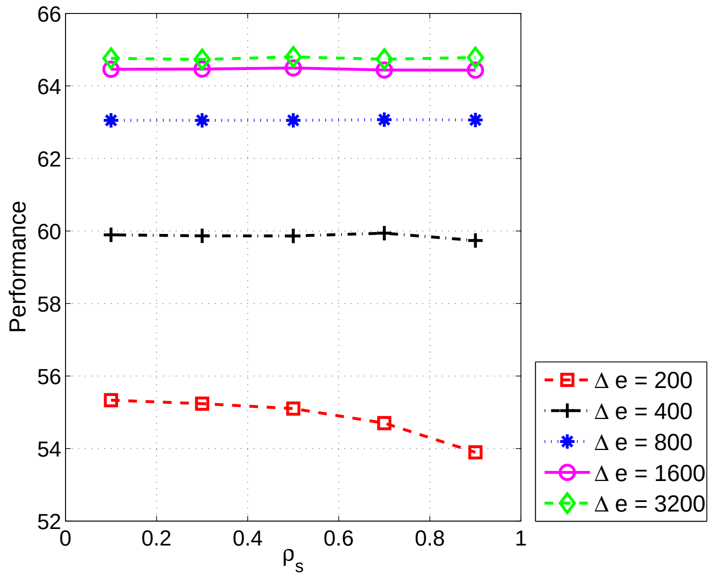


Figure 6.2: The averaged performance measure using Equation 6.9 obtained by DPS with different settings of ρ_s and Δe over 30 runs when maximizing 'average fitness' with the time window $T = 1$ on TP_{14}.

6.3.3.3 Influence of the Parameter ρ_{init}

We fix the setting of ρ_s ($\rho_s = 0.2$) and vary the setting of ρ_{init} ($\rho_{init} \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$) in DPS. From Figure 6.3, we can see that a larger value of ρ_{init} gives a better performance when Δe is small ($\Delta e = 200$). When Δe is getting larger, the impact of ρ_{init} on the performance of DPS becomes negligible. The reason is that when Δe is small there is not enough time for a population to converge to a good solution, and therefore an initial population with a larger convergence, i.e., when ρ_{init} is large, is more likely to find a good solution. On the other hand, when Δe is large, the diversity of an initial population plays a more important role. Besides, in DPS we sample individuals from Gaussian distributions with reasonably set minimal variances, and therefore different settings of ρ_{init} won't result in much difference in the diversity of an initial population, as demonstrated in Figure 6.1. Similar phenomenon can be observed on other test problems when maximizing either 'average fitness' or 'survival time'.

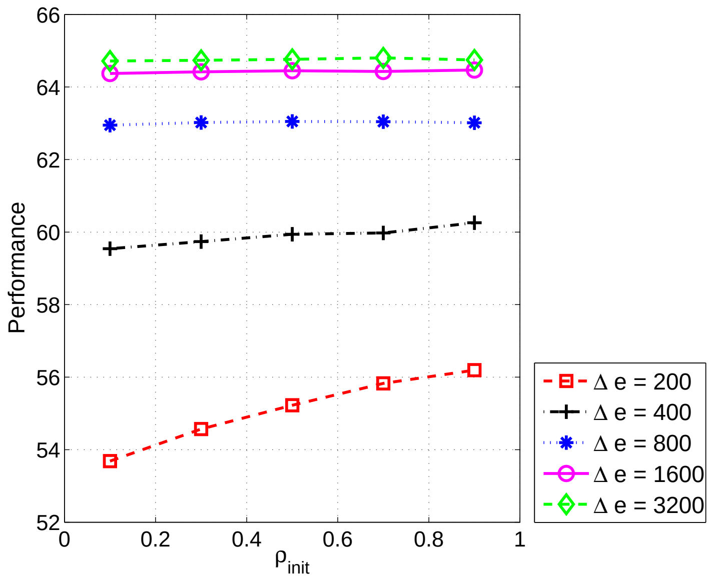


Figure 6.3: The averaged performance measure using Equation 6.9 obtained by DPS with different settings of ρ_{init} and Δe over 30 runs when maximizing 'average fitness' with the time window $T = 1$ on TP_{14}.

6.4 Summary

The main contribution of this chapter is a new dynamic handling strategy, i.e., DPS, which is used to react to environmental changes in ROOT problems by initializing a new population whenever the DFF changes. In comparison to other dynamic handling strategies, DPS builds an explicit time series model on distributions of good solutions over time. Whenever the environment changes, the distributions of good solutions are predicted and used to initialize some individuals in the population. The main advantages of DPS over existing dynamic handling strategies are two-fold. The time series in DPS consists of distributions of good solutions over time, and therefore more information can be learned from the past, compared to the time series consisting of best solutions found over time. Furthermore, an explicit learning model, i.e., the ensemble, is built in DPS to give predictions of the distributions, which enables DPS to deal with more types of dynamics than *implicit prediction strategies*.

We demonstrated the effectiveness of DPS over other strategies from the literature on 18 benchmark problems with different dynamics, which confirms our claim that DPS is able to deal with more types of dynamics and has a significantly better performance under different settings of Δe in most cases. We also showed that DPS, compared to other strategies, can strike a better balance between the diversity and the convergence of an initial population, which explains the success of DPS from a different perspective. Finally, we studied the influences of the time series prediction model and the settings of ρ_s and ρ_{init} on DPS. We found that a more accurate prediction model can benefit DPS and that ρ_s and ρ_{init} generally do not have a big impact on DPS except when Δe is small.

In DPS, we modelled good solutions at a time step using a Gaussian distribution, which is inherently uni-modal. This is a rough approximation of the true distribution where the fitness landscape at a time step is multi-modal. In the future, we would like to first extend the current DPS by employing a Gaussian mixture distribution to model good solutions at a time step. Secondly, we estimated and predicted distributions of solutions that are good in terms of fitness at a time step, and we employed a heuristic to initialize individuals that are hopefully good in terms of the corresponding robustness. It will be interesting to directly estimate and predict distributions of solutions that are good in terms of the corresponding robustness as it is the robustness that we want to maximize in ROOT problems. Finally, it is worth applying the idea of DPS, which is building an explicit time series learning model on distributions, to other types of DOPs or DMOPs.

CHAPTER 7
CONCLUSIONS AND FUTURE WORK

This chapter concludes the thesis, and some future work are discussed with regard to the work presented in this thesis.

7.1 Conclusions

This thesis is dedicated to the research of one type of DOPs, i.e., ROOT problems, and has given an answer to each of the research questions proposed in Section 1.2. To be more specific, the thesis has made the following contributions:

In Chapter 2, research in EDO was briefly reviewed, in which we have identified four major types of DOPs, namely TMO problems, AO problems, DOPs with time-linkage, and ROOT problems. More importantly, the unique feature of ROOT problems, which is finding robust solutions that can be used after environmental changes in DOPs, was made clear in comparison to other three major types of DOPs. Also, similar motivations to ROOT in the literature have been discussed, which further emphasizes the unique feature of ROOT. Moreover, main problems with existing research on ROOT were discussed, which are that a practical definition of ROOT problems is lacking and that the robustness of ROOT solutions is not clearly defined.

In Chapter 3, we proposed a practical problem definition of ROOT, which captures the unique feature of ROOT problems compared to other types of DOPs in EDO. Two robustness definitions, i.e., 'average fitness' and 'survival time', were also proposed, which have been motivated by investigating real world DOPs where it is desirable to have robust solutions that can be used after environmental changes. Assumptions of ROOT problems investigated in this thesis were then made clear in the hope that ROOT can be as general and practical as possible on the one hand and easily formulated on the other hand. Moreover, some discussions with regard to the difficulties of solving ROOT problems from the perspective of an EA were presented.

In Chapter 4, we investigated the design of benchmark problems for the study of ROOT. We argued that it is not sufficient to test an algorithm's ROOT ability on existing DOP benchmarks, and the main reason is that it is difficult, if not impossible, to know the absolute best performance, in terms of either 'average fitness' or 'survival time', in existing benchmarks. The information of the absolute best performance in benchmarks is necessary as it gives the absolute best performance against which an algorithm's performance can be evaluated. Based on this argument, we developed two benchmark problems respectively for the study of maximizing 'average fitness' and 'survival time' in ROOT. Some EDO methods were then tested on our ROOT benchmarks, which demonstrated, for the first time, the gaps between the performance of existing methods and the absolute best performance on our ROOT benchmarks. Moreover, the strengths and weaknesses of existing methods on ROOT problems with different dynamics were discussed.

In Chapter 5, we developed an algorithm framework, called OEL, for solving ROOT problems. OEL features both optimizing and ensemble learning. To be more specific, a population-based search algorithm is employed to search for ROOT solutions based on the corresponding metric (a metric for maximizing 'average fitness' in ROOT and a metric for maximizing 'survival time' in ROOT were developed). In order to calculate the metric, a solution's future fitness is predicted via ensemble learning, which is built based on a solution fitness time series over time. A solution's fitness time series consists of its current fitness evaluated using the current fitness function and its fitness at previous time steps that is estimated by building a global surrogate model using all solutions evaluated at the same previous time step. OEL was then evaluated against existing methods for ROOT on a widely used benchmark in EDO and benchmarks we developed specially for ROOT in Chapter 4. It was shown that OEL achieves significantly better performance than existing methods on ROOT problems when maximizing 'average fitness' or 'survival time' under different dynamics. The better performance of OEL is due to the metrics that are developed specially for maximizing the corresponding robustness and the ensemble learning mechanism that achieves a more accurate prediction of a solution's future fitness than a single model.

An important aspect in solving ROOT problems using EAs is handling environmental changes in ROOT problems. There have been various strategies in EDO for handling environmental changes in different types of DOPs and also DMOPs. In Chapter 6, we developed a novel dynamic handling strategy for ROOT problems. In contrast to existing dynamic handling strategies, our dynamic handling strategy, i.e., DPS, predicts distributions of good solutions and uses the prediction to initialize a promising population whenever there is an environmental change in ROOT. There are two features in DPS that may be advantageous over existing dynamic handling strategies. One feature is that DPS organizes historical evaluation information in the form of distribution of good solutions. The other feature is that DPS employs an explicit learning model (i.e., an ensemble) to learn and predict distributions of good solutions. The effectiveness of DPS over other strategies from the literature was then demonstrated on three types of benchmark problems (a widely used benchmark in EDO and two benchmark problems developed in Chapter 4) with different dynamics and under different budgets of computational resource, which showed that DPS is able to deal with more types of dynamics and has a significantly better performance in most cases. It was also demonstrated that DPS is able to strike a better balance between the diversity and the convergence of an initial population, and this explains the success of DPS over existing dynamic handling strategies from a different perspective.

7.2 Future Work

By finding robust solutions in ROOT, the adaptation cost, which is incurred when adapting a previously implemented solution for a new environment, can be saved as a ROOT solution is aimed to be used for multiple time steps. Yet, the adaptation cost has not been explicitly considered in ROOT, and whether ROOT could strike a better balance between the adaptation cost and the performance than AO, where both the adaptation cost and the performance are optimized, is still an open question. For the future work, it is therefore worth comparing the performance and the adaptation cost when formulating a DOP as a ROOT problem to those when formulating a DOP as a AO problem. Also, we can extend ROOT to multi-objective cases where the adaptation cost is explicitly considered.

It was found in (Nguyen [2011]) that, in some real world DOPs, the constraints of the solution space are also changing over time in addition to the fitness function. Therefore, it is necessary in some cases to account for dynamic constraints in ROOT as well. Dynamic constraints will present new challenges to ROOT as a previously found robust solution could be infeasible after an environmental change. As a result, not only a solution's future fitness needs to be considered in ROOT, but also the solution has to maintain its feasibility after environmental changes. In other words, it is worth developing specific methods for handling dynamic constraints in ROOT in the future.

One important assumption about ROOT problems investigated in this thesis is that time-linkage is not considered. This assumption could be invalid in some cases where there exists some level of time-linkage in which a previously implemented solution influences how the environment changes in the future. Additional techniques are needed to extract the actual dependence of future environments on previously implemented solutions so that time-linkage can be explicitly accounted for in ROOT. The issue of time-linkage has been extensively studied in the research community of reinforcement learning (Sutton and Barto [1998]), and we have recently drawn a connection between EDO and reinforcement learning (Fu et al. [2014a]). It is therefore interesting to extend existing ROOT methods to situations with time-linkage by transferring knowledge from reinforcement learning.

Theoretical analysis of ROOT is also worth pursuing in the future. For instance, it would be interesting to strictly analyse the relationship between the ROOT performance of a standard EA and the accuracy of the metric that is used to guide the EA. The reason is that a solution's robustness in ROOT involves the solution's future fitness, and therefore any metric used in the EAs for ROOT problems is inevitably inaccurate. Also, it would be interesting to know the exact impact of the computational budget Δe at a time step on the ROOT performance of a standard EA given a certain noisy metric. Moreover, rigorous runtime analysis of EAs for ROOT problems under certain assumptions is of great interest and importance to both the academia research of ROOT and its potential real world applications.

Appendices

.1 Proof of Lemma 4.2.1

Remember that x_j and all the centres $c^{i_kj}_{t+k}$, $0 \le k \le T-1$, belong to the interval $[a, b]$. By definition, the MAF of a set of peak functions $\{peak^{i_kj}_{t+k} | 0 \le k \le T-1\}$ is $\max\{\frac{1}{T}\sum_{k=0}^{T-1}(h^{i_kj}_{t+k} - w^{i_kj}_{t+k} * |x_j - c^{i_kj}_{t+k}|) | x_j \in [a, b]\}$. Without loss of generality, suppose all the centres $c^{i_kj}_{t+k}$, $0 \le k \le T-1$, are in an ascending order in the list $(c^{i_0j}_t, c^{i_1j}_{t+1}, ..., c^{i_{T-1}j}_{t+T-1})$. It is easy to verify that $\frac{1}{T}\sum_{k=0}^{T-1}(h^{i_kj}_{t+k} - w^{i_kj}_{t+k} * |x_j - c^{i_kj}_{t+k}|)$ is monotonically increasing for the interval $[a, c^{i_0j}_t]$ and monotonically decreasing for the interval $[c^{i_{T-1}j}_{t+T-1}, b]$. For any of the intervals $[c^{i_{k-1}j}_{t+k-1}, c^{i_kj}_{t+k}]$, $1 \le k \le T-1$, either of the following statements is true: $\sum_{l=0}^{k-1} w^{i_lj}_{t+l} < \sum_{l=k}^{T-1} w^{i_lj}_{t+l}$ and $\sum_{l=0}^{k-1} w^{i_lj}_{t+l} \ge \sum_{l=k}^{T-1} w^{i_lj}_{t+l}$. This means $\frac{1}{T}\sum_{k=0}^{T-1}(h^{i_kj}_{t+k} - w^{i_kj}_{t+k} * |x_j - c^{i_kj}_{t+k}|)$ is monotonically either decreasing or increasing in the interval $[c^{i_{k-1}j}_{t+k-1}, c^{i_kj}_{t+k}]$. Therefore, we have that the MAF for a set of peak functions $\{peak^{i_kj}_{t+k} | 0 \le k \le T-1\}$ can be achieved when x_j takes the value of one of the centres $c^{i_kj}_{t+k}$, $0 \le k \le T-1$.

.2 Proof of Lemma 4.2.2

By definition, the MAF of a set of dimensional functions $\{dim^j_{t+k} | 0 \le k \le T-1\}$ is $\max\{\frac{1}{T}\sum_{k=0}^{k=T-1}\max_{i=1}^{i=m}\{h^{ij}_{t+k} - w^{ij}_{t+k} * |x_j - c^{ij}_{t+k}|\} \mid x_j \in [a, b]\}$. Without loss of generality, suppose the MAF of the set of dimensional functions is achieved at point x^*_j. As a result, there exists a set of i^*_ks, $0 \le k \le T-1$, such that the MAF of the set of dimensional functions equals $\frac{1}{T}\sum_{k=0}^{k=T-1}(h^{i^*_kj}_{t+k} - w^{i^*_kj}_{t+k} * |x^*_j - c^{i^*_kj}_{t+k}|)$. $\frac{1}{T}\sum_{k=0}^{k=T-1}(h^{i^*_kj}_{t+k} - w^{i^*_kj}_{t+k} * |x^*_j - c^{i^*_kj}_{t+k}|)$ is no bigger than $MAF(peak^{i^*_0j}_t, peak^{i^*_1j}_{t+1}, ..., peak^{i^*_{T-1}j}_{t+T-1})$, which is no bigger than $\max\{MAF(peak^{i_0j}_t, peak^{i_1j}_{t+1}, ..., peak^{i_{T-1}j}_{t+T-1}) \mid 1 \le i_k \le m, 0 \le k \le T-1\}$.

On the other hand, without loss of generality, suppose $\max\{MAF(peak^{i_0j}_t, peak^{i_1j}_{t+1}, ..., peak^{i_{T-1}j}_{t+T-1}) \mid 1 \le i_k \le m, 0 \le k \le T-1\}$ is obtained on a set of i'_ks, $0 \le k \le T-1$, such that $\max\{MAF(peak^{i_0j}_t, peak^{i_1j}_{t+1}, ..., peak^{i_{T-1}j}_{t+T-1}) \mid 1 \le i_k \le m, 0 \le k \le T-1\}$ is equal to $MAF(peak^{i'_0j}_t, peak^{i'_1j}_{t+1}, ..., peak^{i'_{T-1}j}_{t+T-1})$. By definition, $MAF(peak^{i'_0j}_t, peak^{i'_1j}_{t+1}, ..., peak^{i'_{T-1}j}_{t+T-1})$ is $\max\{\frac{1}{T}\sum_{k=0}^{k=T-1}(h^{i'_kj}_{t+k} - w^{i'_kj}_{t+k} * |x_j - c^{i'_kj}_{t+k}|) \mid x_j \in [a, b]\}$, which is no bigger than $\max\{\frac{1}{T}\sum_{k=0}^{k=T-1}\max_{i=1}^{i=m}\{h^{ij}_{t+k} -$

$w_{t+k}^{ij} * |x_j - c_{t+k}^{ij}|\} \mid x_j \in [a,b]\}$, i.e., $MAF(dim_t^j, dim_{t+1}^j, ..., dim_{t+T-1}^j)$.

Therefore, we have Lemma 4.2.2 proved.

.3 Proof of Theorem 4.2.1

By definition, the MAF of a set of fitness functions $\{f_{t+k}^a | 0 \leq k \leq T-1\}$ is $\max\{\frac{1}{T}\sum_{k=0}^{T-1}\frac{1}{d}\sum_{j=1}^{d}\max_{i=1}^{i=m}\{h_{t+k}^{ij} - w_{t+k}^{ij} * |x_j - c_{t+k}^{ij}|\} \mid x_j \in [a,b], 1 \leq j \leq d\}$. By arithmetic operation and since d is finite, we have:

$$\max\{\frac{1}{T}\sum_{k=0}^{T-1}\frac{1}{d}\sum_{j=1}^{d}\max_{i=1}^{i=m}\{h_{t+k}^{ij} - w_{t+k}^{ij} * |x_j - c_{t+k}^{ij}|\} \mid x_j \in [a,b], 1 \leq j \leq d\} =$$

$$\frac{1}{d}\sum_{j=1}^{d}\max\{\frac{1}{T}\sum_{k=0}^{T-1}\max_{i=1}^{i=m}\{h_{t+k}^{ij} - w_{t+k}^{ij} * |x_j - c_{t+k}^{ij}|\} \mid x_j \in [a,b]\} =$$

$$\frac{1}{d}\sum_{j=1}^{d} MAF(dim_t^j, dim_{t+1}^j, ..., dim_{t+T-1}^j).$$

Based on Lemma 4.2.2, we have $MAF(dim_t^j, dim_{t+1}^j, ..., dim_{t+T-1}^j)$ equal to $\max\{MAF(peak_t^{i_0 j}, peak_{t+1}^{i_1 j}, ..., peak_{t+T-1}^{i_{T-1} j}) \mid 1 \leq i_k \leq m, 0 \leq k \leq T-1\}$. Therefore, we have Theorem 4.2.1 proved.

.4 Proof of Lemma 4.2.3

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

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

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

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

Therefore, we have Lemma 4.2.3 proved.

.5 Proof of Lemma 4.2.4

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

On the other hand, without loss of generality, suppose $\max\{MIF(peak_t^{i_0 j},$

$peak_{t+1}^{i_1 j}, ..., peak_{t+L-1}^{i_{L-1} j}) \mid 1 \le i_k \le m, 0 \le k \le L-1\}$ is obtained on a set of i'_ks, $0 \le k \le L-1$, such that $\max\{MIF(peak_t^{i_0 j}, peak_{t+1}^{i_1 j}, ..., peak_{t+L-1}^{i_{L-1} j}) \mid 1 \le i_k \le m, 0 \le k \le L-1\}$ equals $MIF(peak_t^{i'_0 j}, peak_{t+1}^{i'_1 j}, ..., peak_{t+L-1}^{i'_{L-1} j})$. By definition, $MIF(peak_t^{i'_0 j}, peak_{t+1}^{i'_1 j}, ..., peak_{t+L-1}^{i'_{L-1} j})$ is $\max\{\min_{k=0}^{k=L-1}(h_{t+k}^{i'_k j} - w_{t+k}^{i'_k j} * |x_j - c_{t+k}^{i'_k j}|) \mid x_j \in [a,b]\}$, which is no bigger than $\max\{\min_{k=0}^{k=L-1} \max_{i=1}^{i=m}\{h_{t+k}^{ij} - w_{t+k}^{ij} * |x_j - c_{t+k}^{ij}|\} \mid x_j \in [a,b]\}$.

Therefore, we have Lemma 4.2.4 proved.

.6 Proof of Theorem 4.2.2

For any dimension j, $1 \le j \le d$, $MIF(f_t^s, f_{t+1}^s, ..., f_{t+L-1}^s) \le MIF(dim_t^j, dim_{t+1}^j, ..., dim_{t+L-1}^j)$, since $f_{t+k}^s \le dim_{t+k}^j, \forall k, 0 \le k \le T-1$. Supposing $MIF(dim_t^{j^*}, dim_{t+1}^{j^*}, ..., dim_{t+L-1}^{j^*}) = \min_{j=1}^{d} MIF(dim_t^j, dim_{t+1}^j, ..., dim_{t+L-1}^j)$, we have $MIF(f_t^s, f_{t+1}^s, ..., f_{t+L-1}^s) \le MIF(dim_t^{j^*}, dim_{t+1}^{j^*}, ..., dim_{t+L-1}^{j^*})$.

On the other hand, suppose for any dimension j, $1 \le j \le d$, $MIF(dim_t^j, dim_{t+1}^j, ..., dim_{t+L-1}^j)$ achieves its maximal at point x_j^*: $MIF(dim_t^j, dim_{t+1}^j, ..., dim_{t+L-1}^j) = min\{dim_{t+i}^j(x_j^*) | 0 \le i \le L-1\}$. By denoting $\mathbf{x}^* = (x_1^*, x_2^*, ..., x_d^*)$, we have $min\{f_{t+i}^s(\mathbf{x}^*) | 0 \le i \le L-1\} = MIF(dim_t^{j^*}, dim_{t+1}^{j^*}, ..., dim_{t+L-1}^{j^*})$. This means $MIF(f_t^s, f_{t+1}^s, ..., f_{t+L-1}^s) \ge min\{f_{t+i}^s(\mathbf{x}^*) | 0 \le i \le L-1\} = MIF(dim_t^{j^*}, dim_{t+1}^{j^*}, ..., dim_{t+L-1}^{j^*})$.

Therefore, we have $MIF(f_t^s, f_{t+1}^s, ..., f_{t+L-1}^s)$ equal to $MIF(dim_t^{j^*}, dim_{t+1}^{j^*}, ..., dim_{t+L-1}^{j^*})$.

Based on Lemma 4.2.4, we have $MIF(dim_t^j, dim_{t+1}^j, ..., dim_{t+L-1}^j)$ equal to $\max\{MIF(peak_t^{i_0 j}, peak_{t+1}^{i_1 j}, ..., peak_{t+L-1}^{i_{L-1} j}) \mid 1 \le i_k \le m, 0 \le k \le L-1\}$. Therefore, we have Theorem 4.2.2 proved.

.7 Surrogate Model for Fitness Approximation

The surrogate model used in this thesis for estimating solution's past fitness is RBFN of the form:

$$f(\mathbf{x}) = \sum_{i=1}^{n_c} \eta_i * K_i(\mathbf{x}, \mathbf{x}_i), \tag{1}$$

where we use the normalized Gaussian radial basis function:

$$K_i(\mathbf{x}, \mathbf{x}_i) = \frac{exp\{(\mathbf{x} - \mathbf{x}_i)^T * \mathbf{C}_i^{-1} * (\mathbf{x} - \mathbf{x}_i)\}}{\sum\limits_{i=1}^{n_c} exp\{(\mathbf{x} - \mathbf{x}_i)^T * \mathbf{C}_i^{-1} * (\mathbf{x} - \mathbf{x}_i)\}}, \tag{2}$$

where $\mathbf{C}_i$ is the covariance matrix for the ith basis function. n_c number of centres, $\mathbf{x}_i (1 \leq i \leq n_c)$, are selected using the K-means algorithm, and the covariance matrix for each cluster is determined by fitting a multivariate Gaussian distribution within each cluster. In this thesis, we consider the diagonal covariance matrix and estimate the variance dimension by dimension (the minimum standard deviation for each dimension is set 0.1). Finally, a least square solution is obtained for the weights η_i, $1 \leq i \leq n_c$. The RBFN has been implemented using the Weka software (Hall et al. [2009]).

.8 Learning Models for Time Series Prediction

Given a time series of length L: $(y_1, y_2, ...y_L)$, the prediction task is to predict future values of y: y_{L+i}, $i \geq 1$. All the following learning models have been implemented using the Weka software (Hall et al. [2009]).

For AR model, a training data set is firstly formed based on $(y_1, y_2, ..., y_L)$ with an embedding size of m: $\{(y_i, y_{i+1}, ...y_{i+m})\}$, $1 \leq i \leq L - m$, considering Takens's theorem (Takens [1981]) about discrete time dynamical system. The first m data points in each instance are treated as the input, and the last data point is the desired output. To determine the coefficients η in AR model:

$$y_t = \sum_{i=1}^{m} \eta_i * y_{t-i} + \varepsilon_t, \tag{3}$$

where ε_t is the Gaussian white noise with variance σ^2, a least square solution of η is obtained on the training data set $\{(y_i, y_{i+1}, ...y_{i+m})\}$, $1 \leq i \leq L - m$. After that, prediction about future values of y is made as:

$$\hat{y}_{L+j} = \sum_{i=1}^{m} \eta_i * y_{L+j-i}, \quad j \geq 1, \tag{4}$$

where $\hat{y}_{L+1}$ is firstly predicted and then used for the prediction of $\hat{y}_{L+2}$. The process is repeated for prediction of later future values of y. The variance σ^2 in AR model is estimated as follows after the determination of η:

$$\hat{\sigma}^2_{AR} = \frac{1}{L-m} \sum_{i=m+1}^{L} (y_i - \sum_{j=1}^{m} \eta_j * y_{i-j})^2. \tag{5}$$

In NN model, a training data set is formed with an embedding size of m as what is done in AR. The basic idea in NN is to use the average output of k nearest neighbours in input space as the prediction. The NN prediction takes the form as:

$$y_t = \frac{1}{n} \sum_{i=1}^{n} y_{t_i} + \varepsilon_t, \tag{6}$$

where n is the neighbourhood size and ε_t is the Gaussian white noise with variance σ^2. y_{t_i} the ith nearest neighbour of y_t in terms of Euclidean distance in input space: $d(y_t, y_{t_i}) = \sqrt{\sum_{j=1}^{m} (y_{t-j} - y_{t_i - j})^2}$. Similar as in AR, $\hat{y}_{L+1}$ is firstly predicted in NN and then used for the prediction of $\hat{y}_{L+2}$. The process is repeated for prediction of later future values of y. The variance σ^2 in NN model is estimated as follows:

$$\hat{\sigma}^2_{NN} = \frac{1}{L-m} \sum_{i=m+1}^{L} (y_i - \frac{1}{n} * \sum_{j=1}^{n} y_{i_j})^2. \tag{7}$$

A model with a constant mean b is assumed in MA model:

$$y_t = b + \varepsilon_t, \tag{8}$$

where ε_t is the Gaussian white noise with variance σ^2. The parameter b is estimated by taking the latest n data points in the time series:

$$b = \frac{1}{n} \sum_{i=1}^{n} y_{L+1-i}. \tag{9}$$

The variance σ^2 is estimated within the latest n data points after b has been estimated:

$$\hat{\sigma}^2_{MA} = \frac{1}{n} \sum_{i=1}^{n} (y_{L+1-i} - b)^2. \tag{10}$$

Therefore, future values of y are all predicted as having a mean of b and a variance of $\hat{\sigma}^2_{MA}$.

The estimation of variance of the Gaussian white noise for a future predicted y by the ensemble in OEL is given by:

$$\hat{\sigma}^2_{OEL} = \frac{1}{k^2} \sum_{i=1}^{k} \hat{\sigma}^2_i, \tag{11}$$

where $\hat{\sigma}^2_i$ is the variance estimation given by the ith learner in the ensemble, and k is the number of learners in the ensemble.

List of References

H. Akaike. Fitting autoregressive models for prediction. *Annals of the institute of Statistical Mathematics*, 21(1):243–247, 1969. 39

J. A. D. Atkin, E. K. Burke, J. S. Greenwood, and D. Reeson. On-line decision support for take-off runway scheduling with uncertain taxi times at london heathrow airport. *Journal of Scheduling*, 11(5):323–346, 2008. 32

G. J. Barlow and S. F. Smith. A memory enhanced evolutionary algorithm for dynamic scheduling problems. In *Applications of Evolutionary Computing*, pages 606–615. Springer, 2008. 2

C. N. Bendtsen and T. Krink. Dynamic memory model for non-stationary optimisation. In *Proc. of the 2002 Congress on Evol. Comput*, pages 145–150, 2002. 17

H. G. Beyer and B. Sendhoff. Robust optimisation-a comprehensive survey. *Computer methods in applied mechanics and engineering*, 196(33-34):3190–3218, 2007. ISSN 0045-7825. 29, 30

T. Blackwell and J. Branke. Multiswarms, exclusion, and anti-convergence in dynamic environments. *Evolutionary Computation, IEEE Transactions on*, 10 (4):459–472, 2006. 15, 16

T. Blackwell, J. Branke, and X. Li. Particle swarms for dynamic optimisation problems. *Swarm Intelligence*, pages 193–217, 2008. 12, 15

P. A. N. Bosman. Learning, anticipation and time-deception in evolutionary online dynamic optimisation. In *Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 39–47. ACM, 2005. 12, 24, 25, 26, 66

P. A. N. Bosman and H. La Poutre. Learning and anticipation in online dynamic optimisation with evolutionary algorithms: the stochastic case. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1165–1172. ACM, 2007. 24, 25, 26, 66

G. E. P. Box and G. M. Jenkins. *Time series analysis: forecasting and control*. Prentice Hall PTR, 1994. 40

G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 3rd edition, 1994. ISBN 0130607746. 59, 78, 103

J. Branke. Memory enhanced evolutionary algorithms for changing optimisation problems. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 1999. 11, 12, 17, 18, 43, 44, 45, 48, 79, 99, 108, 113

J. Branke. *Evolutionary Optimisation in Dynamic Environments*. Kluwer Academic Publishers, Norwell, MA, USA, 2001. ISBN 0792376315. 2, 3, 5

J. Branke. *Evolutionary optimisation in dynamic environments*, volume 3. Kluwer Academic Pub, 2002. 11, 22, 23

J. Branke and D. Mattfeld. Anticipation in dynamic optimisation: The scheduling case. In *Parallel Problem Solving from Nature–PPSN VI*, pages 253–262. Springer, 2000. 19, 20

J. Branke, T. Kaußler, C. Schmidt, and H. Schmeck. A multi-population approach to dynamic optimisation problems. *Adaptive computing in design and manufacturing*, 2000:299–308, 2000. 18

L. T. Bui, M. H. Nguyen, J. Branke, and H. A. Abbass. Tackling dynamic problems with multiobjective evolutionary algorithms. *Multiobjective Problem Solving from Nature*, pages 77–91, 2008. 15, 16

L. T. Bui, Z. Michalewicz, E. Parkinson, and M. B. Abello. Adaptation in dynamic environments: a case study in mission planning. *Evolutionary Com-*

putation, IEEE Transactions on, 16(2):190–209, 2012. 2, 3, 12, 22, 23, 32, 100

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. 59, 74, 82, 111, 112

H. G. Cobb. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical report, DTIC Document, 1990. 14, 43, 99, 100

S. Droste. Analysis of the (1+ 1) ea for a dynamically changing onemax-variant. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 1, pages 55–60. IEEE, 2002. 11

S. Droste. Analysis of the (1+ 1) ea for a dynamically bitwise changing onemax. In *Genetic and Evolutionary ComputationGECCO 2003*, pages 202–202. Springer, 2003. 11

J. Eggermont, T. Lenaerts, S. Poyhonen, and A. Termier. Raising the dead: Extending evolutionary algorithms with a case-based memory. In *Genetic Programming*, pages 280–290. Springer, 2001. 17, 18

J. D. Farmer and J. J. Sidorowich. Predicting chaotic time series. *Physical review letters*, 59(8):845, 1987. 78, 103

H. Fu, H. Chen, B. Sendhoff, K. Tang, and X. Yao. Distribution prediction in robust optimisation over time. *submitted to Evolutionary Computation, IEEE Transactions on*, a.

H. Fu, L. L. Minku, B. Sendhoff, K. Tang, and X. Yao. Optimizing and learning in robust optimisation over time. *submitted to Cybernetics IEEE Transactions on*, b.

H. Fu, B. Sendhoff, K. Tang, and X. Yao. Robust optimisation over time: Problem difficulties and benchmark problems. *Evolutionary Computation, IEEE Transactions on*, c.

H. Fu, B. Sendhoff, K. Tang, and X. Yao. Characterizing environmental changes in robust optimisation over time. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012. 13, 27

H. Fu, B. Sendhoff, K. Tang, and X. Yao. Finding robust solutions to dynamic optimisation problems. In *Applications of Evolutionary Computation*, pages 616–625. Springer, 2013. 13, 27, 34, 59, 62, 68, 69, 72

H. Fu, P. R. Lewis, B. Sendhoff, K. Tang, and X. Yao. What are dynamic optimisation problems? In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC)*. IEEE Press, 2014a. in press. 128

H. Fu, P. R. Lewis, and X. Yao. A q-learning based evolutionary algorithm for sequential decision making problems. *In Proceedings of the Workshop "In Search of Synergies between Reinforcement Learning and Evolutionary Computation" at the 13th International Conference on Parallel Problem Solving from Nature (PPSN)*, 2014b. in press.

D. Gianazza. Forecasting workload and airspace configuration with neural networks and tree search methods. *Artificial intelligence*, 174(7):530–549, 2010. 32

D. E. Goldberg and R. E. Smith. Nonstationary function optimisation using genetic algorithm with dominance and diploidy. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 59–68. L. Erlbaum Associates Inc., 1987. 16

J. J. Grefenstette. Genetic algorithms for changing environments. *Parallel problem solving from nature*, 2:137–144, 1992. 15

Y. Guo, M. Chen, H. Fu, and Y. Liu. Find robust solutions over time by two-layer multi-objective optimisation method. In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC)*. IEEE Press, 2014. in press.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009. 135

H. Handa. Fitness function for finding out robust solutions on time-varying functions. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1195–1200. ACM, 2006. ISBN 1595931864. 28, 32

H. Handa, L. Chapman, and X. Yao. Dynamic salting route optimisation using evolutionary computation. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 158–165. IEEE, 2005. 3, 32

I. Hatzakis and D. Wallace. Dynamic multi-objective optimisation with evolutionary algorithms: a forward-looking approach. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1201–1208. ACM, 2006. 19, 20, 99, 100, 103, 113

S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979. 85

X. Hu and R. C. Eberhart. Adaptive particle swarm optimisation: detection and response to dynamic systems. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pages 1666–1670. IEEE, 2002. 14, 99, 100, 112

Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 9(1):3–12, 2005. ISSN 1432-7643. 40

Y. Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011. 68

Y. Jin and J. Branke. Evolutionary optimisation in uncertain environments-a survey. *Evolutionary Computation, IEEE Transactions on*, 9(3):303–317, 2005. ISSN 1089-778X. 2, 3, 11, 29, 40

Y. Jin and B. Sendhoff. Constructing dynamic optimisation test problems using the multi-objective optimisation concept. *Applications of Evolutionary Computing*, pages 525–536, 2004. 11, 43, 45, 46, 79, 108

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. 3, 13, 27, 59, 67, 68, 69, 76, 84, 112

A. Larsen and O. B. Madsen. *The dynamic vehicle routing problem*. PhD thesis, Technical University of DenmarkDanmarks Tekniske Universitet, Department of TransportInstitut for Transport, Logistics & ITSLogistik & ITS, 2000. 32

S. C. H. Leung, S. O. S. Tsang, W. L. Ng, and Y. Wu. A robust optimisation model for multi-site production planning problem in an uncertain environment. *European Journal of Operational Research*, 181(1):224–238, 2007. 28

J. Lewis, E. Hart, and G. Ritchie. A comparison of dominance mechanisms and simple mutation on non-stationary problems. In *Parallel Problem Solving from Nature–PPSN V*, pages 139–148. Springer, 1998. 16, 43

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. 15

C. Li, S. Yang, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H. G. Beyer, and P. N. Suganthan. Benchmark generator for CEC 2009 competition on dynamic optimisation. *University of Leicester, University of Birmingham, Nanyang Technological University, Tech. Rep*, 2008. 43, 45, 51, 52, 79, 80, 108, 109

J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 14. California, USA, 1967. 76, 111

M. Mavrovouniotis and S. Yang. Memory-based immigrants for ant colony optimisation in changing environments. In *Applications of Evolutionary Computation*, pages 324–333. Springer, 2011. 17

Z. Michalewicz, M. Schmidt, M. Michalewicz, and C. Chiriac. Adaptive business intelligence: three case studies. In *Evolutionary Computation in Dynamic and Uncertain Environments*, pages 179–196. Springer, 2007. 2

N. Mori, H. Kita, and Y. Nishikawa. Adaptation to a changing environment by means of the feedback thermodynamical genetic algorithm. In *Parallel Problem Solving from Nature-PPSN V*, pages 149–158. Springer, 1998. 15, 17

R. Morrison. Performance measurement in dynamic environments. In *GECCO workshop on evolutionary algorithms for dynamic optimisation problems*, pages 5–8. Citeseer, 2003. 11

R. W. Morrison and K. A. De Jong. A test problem generator for non-stationary environments. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, pages 2047–2053 Vol. 3. IEEE, 1999. 11, 43, 45, 79, 108, 109

K. P. Ng and K. C. Wong. A new diploid scheme and dominance change mechanism for non-stationary function optimisation. In *Proceedings of the 6th international conference on genetic algorithms*, pages 159–166. Morgan Kaufmann Publishers Inc., 1995. 16

T. T. Nguyen. *Continuous dynamic optimisation using evolutionary algorithms*. PhD thesis, University of Birmingham, 2011. 11, 37, 53, 66, 128

T. T. Nguyen and X. Yao. Benchmarking and solving dynamic constrained problems. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 690–697. IEEE, 2009a. 11, 12, 15, 43

T. T. Nguyen and X. Yao. Dynamic time-linkage problems revisited. In *Applications of Evolutionary Computing*, volume 5484 of *Lecture Notes in Computer Science*, pages 735–744. Springer, 2009b. 24, 25, 26

T. T. Nguyen and X. Yao. Dynamic time-linkage evolutionary optimisation: Definitions and potential solutions. In *Metaheuristics for Dynamic Optimisation*, pages 371–395. Springer, 2013. 12, 24, 25, 26, 66

T. T. Nguyen, S. Yang, and J. Branke. Evolutionary dynamic optimisation: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6:1–24, 2012a. 2, 3, 11, 79, 99, 108

T. T. Nguyen, Z. Yang, and S. Bonsall. Dynamic time-linkage problems-the challenges. In *Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2012 IEEE RIVF International Conference on*, pages 1–6. IEEE, 2012b. 24, 25, 26

P. Nuzzo, A. Puggelli, S. A. Seshia, and A. Sangiovanni-Vincentelli. Calcs: Smt solving for non-linear convex constraints. In *Formal Methods in Computer-Aided Design (FMCAD), 2010*, pages 71–79. IEEE, 2010. 47

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. ISSN 1089-778X. 29

D. Parrott and X. Li. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *Evolutionary Computation, IEEE Transactions on*, 10(4):440–458, 2006. 15, 18, 19

M. P. Perrone and L. N. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. Technical report, DTIC Document, 1992. 70, 71

S. Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Transactions on Computational Logic (TOCL)*, 7(4):723–748, 2006. 46

H. Richter. Memory design for constrained dynamic optimisation problems. In *Applications of Evolutionary Computation*, pages 552–561. Springer, 2010. 17

H. Richter and S. Yang. Memory based on abstraction for dynamic fitness functions. In *Applications of Evolutionary Computing*, pages 596–605. Springer, 2008. 17

P. Rohlfshagen and X. Yao. Dynamic combinatorial optimisation problems: an analysis of the subset sum problem. *Soft Computing*, 15(9):1723–1734, 2011. 12

P. Rohlfshagen, P. K. Lehre, and X. Yao. Dynamic evolutionary optimisation: an analysis of frequency and magnitude of change. In *Proceedings of the 11th*

Annual conference on Genetic and evolutionary computation, pages 1713–1720. ACM, 2009. 11

C. Rossi, M. Abderrahim, and J. C. Díaz. Tracking moving optima using kalman-based predictions. *Evolutionary computation*, 16(1):1–30, 2008. 3, 19, 20, 32, 99

J. E. Rowe. Finding attractors for periodic fitness functions. In *Genetic and Evolutionary Computation Conference*, pages 557–563, 1999. 11

J. E. Rowe. Cyclic attractors and quasispecies adaptability. In *Theoretical aspects of evolutionary computing*, pages 251–259. Springer-Verlag, 2001. 11

S. Salomon, G. Avigad, P. J. Fleming, and R. C. Purshouse. Active robust optimisation-enhancing robustness to uncertain environments. Technical Report 1040, Department of Automatic Control and Systems Engineering, University of Sheffield, November 1, 2013. 28

A. Simoes and E. Costa. Improving memory's usage in evolutionary algorithms for changing environments. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 276–283. IEEE, 2007. 18

A. Simões and E. Costa. Evolutionary algorithms for dynamic environments: Prediction using linear regression and markov chains. *Parallel Problem Solving from Nature–PPSN X*, pages 306–315, 2008. 17

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. 19, 20, 21, 99, 100

A. R. Solow and S. Polasky. Measuring biological diversity. *Environmental and Ecological Statistics*, 1(2):95–103, 1994. 119

P. D. Stroud. Kalman-extended genetic algorithm for search in nonstationary environments with noisy fitness evaluations. *Evolutionary Computation, IEEE Transactions on*, 5(1):66–77, 2001. 2, 32

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998. 128

F. Takens. Detecting strange attractors in turbulence. In *Dynamical systems and turbulence, Warwick 1980*, pages 366–381. Springer, 1981. 135

E. K. Tang, P. N. Suganthan, and X. Yao. An analysis of diversity measures. *Machine Learning*, 65(1):247–271, 2006. 71

R. Tinós and S. Yang. Continuous dynamic problem generators for evolutionary algorithms. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 236–243. IEEE, 2007. 43

K. Trojanowski and Z. Michalewicz. Searching for optima in non-stationary environments. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 1999. 18

R. K. Ursem. Multinational GAs: Multimodal optimisation techniques in dynamic environments. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 19–26, 2000. 18, 19

R. K. Ursem, T. Krink, M. T. Jensen, and Z. Michalewicz. Analysis and modeling of control tasks in dynamic systems. *Evolutionary Computation, IEEE Transactions on*, 6(4):378–389, 2002. 2, 3, 32, 43

A. S. Uyar and A. E. Harmanci. A new population based adaptive domination change mechanism for diploid genetic algorithms in dynamic environments. *Soft Computing*, 9(11):803–814, 2005. 17

S. Van de Vonder, E. Demeulemeester, and W. Herroelen. A classification of predictive-reactive project scheduling procedures. *Journal of Scheduling*, 10 (3):195–207, 2007. 12, 22

S. Van de Vonder, E. Demeulemeester, and W. Herroelen. Proactive heuristic procedures for robust project scheduling: An experimental analysis. *European Journal of Operational Research*, 189(3):723–733, 2008. 12, 22, 28

J. Van Hemert, C. Van Hoyweghen, E. Lukschandl, and K. Verbeeck. A futurist approach to dynamic environments. In *GECCO EvoDOP Workshop*, pages 35–38. Citeseer, 2001. 19, 20

F. Vavak, K. Jukes, and T.C. Fogarty. Learning the local search range for genetic optimisation in nonstationary environments. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 355–360. IEEE, 1997. 14

K. Weicker. Performance measures for dynamic environments. *Parallel Problem Solving from Nature–PPSN VII*, pages 64–73, 2002. 11

D. E. Wilkins, S. F. Smith, L. A. Kramer, T. J. Lee, and T. W. Rauenbusch. Airlift mission monitoring and dynamic rescheduling. *Engineering Applications of Artificial Intelligence*, 21(2):141–155, 2008. 32

Y. G. Woldesenbet and G. G. Yen. Dynamic evolutionary algorithm with variable relocation. *Evolutionary Computation, IEEE Transactions on*, 13(3):500–513, 2009. 12, 14, 18, 99, 100

S. Yang. Non-stationary problem optimisation using the primal-dual genetic algorithm. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 3, pages 2246–2253. IEEE, 2003. 43

S. Yang. Memory-based immigrants for genetic algorithms in dynamic environments. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1115–1122. ACM, 2005. 17, 18

S. Yang. Associative memory scheme for genetic algorithms in dynamic environments. In *Applications of evolutionary computing*, pages 788–799. Springer, 2006a. 17

S. Yang. On the design of diploid genetic algorithms for problem optimisation in dynamic environments. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1362–1369. IEEE, 2006b. 17

S. Yang and C. Li. A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. *Evolutionary Computation, IEEE Transactions on*, 14(6):959–974, 2010. 12, 15, 18, 19, 112

S. Yang and X. Yao. Experimental study on population-based incremental learning algorithms for dynamic optimisation problems. *Soft Computing*, 9(11): 815–834, 2005. 15

S. Yang and X. Yao. Population-based incremental learning with associative memory for dynamic environments. *Evolutionary Computation, IEEE Transactions on*, 12(5):542–561, 2008. 17, 18, 99

S. Yang, H. Cheng, and F. Wang. Genetic algorithms with immigrants and memory schemes for dynamic shortest path routing problems in mobile ad hoc networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 40(1):52–63, 2010. 2, 113

X. Yu, Y. Jin, K. Tang, and X. Yao. Robust optimisation over time–A new perspective on dynamic optimisation problems. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–6. IEEE, 2010. 3, 12, 26, 27, 67

A. Zhou, Y. Jin, Q. Zhang, B. Sendhoff, and E. Tsang. Prediction-based population re-initialization for evolutionary dynamic multi-objective optimisation. In *Evolutionary Multi-Criterion Optimisation*, pages 832–846. Springer, 2007. 19

A. Zhou, B. Qu, H. Li, S. Zhao, P. N. Suganthan, and Q. Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49, 2011. 12

A. Zhou, Y. Jin, and Q. Zhang. A population prediction strategy for evolutionary dynamic multiobjective optimisation. *Cybernetics, IEEE Transactions on*, PP (99):1–1, 2013. ISSN 2168-2267. doi: 10.1109/TCYB.2013.2245892. 99, 100, 113