UNIVERSITY OF ADELAIDE

DOCTORAL THESIS

Differential Evolution for Dynamic Constrained Continuous Optimisation

Author:
MARYAM HASANI SHOREH

Supervisor:
Prof. FRANK NEUMANN

Co-Supervisors:
Dr. MARÍA-YANELI AMECA-ALDUCIN
Dr. WANRU GAO

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Optimization and Logistics
School of Computer Science

December 7, 2020

Declaration of Authorship

I, MARYAM HASANI SHOREH, declare that this thesis titled, "Differential Evolution for Dynamic Constrained Continuous Optimisation" and the work presented in it are my own. I confirm that:

- I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.
- I acknowledge that copyright of published works contained within this thesis resides with the copyright holder(s) of those works.
- I give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.
- I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Signed: 

Date: 07/12/2020

Maryam Hasani Shoreh

UNIVERSITY OF ADELAIDE

Abstract

Faculty of Engineering, Computer and Mathematical Sciences

School of Computer Science

Doctor of Philosophy

Differential Evolution for Dynamic Constrained Continuous Optimisation

by MARYAM HASANI SHOREH

In this thesis, we choose the evolutionary dynamic optimisation methodology to tackle dynamic constrained problems. Dynamic constrained problems represent a common class of optimisation that occur in many real-world scenarios. Evolutionary algorithms are often considered very general search heuristics. Their main advantages (in comparison to problem-specific search methods) are their robustness, flexibility and extensibility, as well as the fact that almost no domain knowledge is required for their implementation and application.

Our research is focused on the following areas. In the first part of the thesis, we modify common constraint handling techniques from static domains to suit dynamic environments. We investigate the deficiencies of such techniques and the potential of each method based on the change characteristics of the environment. In the second part, we propose a framework to create benchmarks, since we have observed a lack of benchmarks to evaluate algorithms in dynamic continuous optimisation. Third, we carry out an exhaustive empirical study of diversity mechanisms applied to solve dynamic constrained optimisation problems. Finally, we investigate the integration of a neural network into the evolution process and analyse it's effectiveness compared to that of popular diversity mechanisms. We address the possibility of integrating such mechanisms with a neural network approach in order to improve the results.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Frank Neumann for the consistent support of my Ph.D. study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. Sincerely, I could not have imagined having a better advisor and mentor for my Ph.D. study.

Besides my advisor, I would like to express my gratitude and appreciation for my co-supervisors: Dr. Maria-Yaneli Ameca-Alducin, and Dr. Wanru Gao, for their insightful comments and encouragement. Especial thanks to Yaneli that not only was my co-supervisor but also was my close friend that listened to my concerns from time to time.

I would like also to thank all the group members in the optimisation and logistics group, particularly, Markus Wagner and Bradly Alexander. It has been a great experience to work in this research group sharing ideas in both research and daily life.

My sincere thanks also goes to Renato Hermoza Aragones, my best friend and the co-author of the last part of my research to his greatest knowledge and insights.

Last but not least, I would like to thank my parents for their patience and support and my brothers: Reza and Morteza. Besides my family, I am also grateful to my friends for supporting me spiritually throughout my Ph.D. in both life and research.

Contents

Declaration of Authorship iii

Abstract vii

Acknowledgements ix

1 Introduction 1

2 Dynamic Constrained Continuous Optimisation 7
- 2.1 Introduction . . . 7
- 2.2 Specifications of the dynamic environments . . . 8
- 2.3 Problem statement . . . 9
- 2.4 Evolutionary algorithms for dynamic problems: state of the art 10
  - 2.4.1 Diversity introducing . . . 10
  - 2.4.2 Maintaining diversity . . . 11
  - 2.4.3 Memory-based . . . 13
  - 2.4.4 Prediction-based . . . 15
  - 2.4.5 Multi-population-based . . . 17
- 2.5 Benchmarks in dynamic optimisation . . . 19
- 2.6 Performance measures . . . 22

3 Evolutionary Dynamic Optimisation 27
- 3.1 Introduction . . . 27
- 3.2 Differential evolution . . . 28
- 3.3 Change detection mechanisms . . . 29
  - 3.3.1 Re-evaluation of solutions . . . 29
  - 3.3.2 Error calculation . . . 29
- 3.4 Change reaction mechanisms . . . 30
  - 3.4.1 Diversity promoting techniques . . . 30
    - 3.4.1.1 Chaos local search . . . 30
    - 3.4.1.2 Crowding . . . 30
    - 3.4.1.3 Fitness diversity . . . 31
    - 3.4.1.4 No diversity mechanism . . . 31

3.4.1.5 Opposition . . . 31
3.4.1.6 Random immigrants . . . 31
3.4.1.7 Restart population . . . 32
3.4.1.8 Hyper-mutation . . . 32
3.4.2 Neural networks . . . 32
3.5 Constraint handling techniques . . . 35
3.5.1 Penalty functions . . . 35
3.5.2 Feasibility rules . . . 36
3.5.3 ϵ-constrained . . . 36
3.5.4 Stochastic ranking . . . 38
3.5.5 Repair methods . . . 38
3.5.5.1 Reference-based repair method . . . 39
3.5.5.2 Offspring-repair method . . . 40
3.5.5.3 Mutant-repair method . . . 40
3.5.5.4 Gradient-based repair method . . . 41

4 Constraint Handling Techniques 43
4.1 Introduction . . . 43
4.2 Standard constraint handling techniques . . . 44
4.2.1 Experimental design . . . 45
4.2.2 Experimental analysis . . . 46
4.2.2.1 Analysis I: performance measure . . . 47
4.2.2.2 Analysis II: behaviour measures . . . 48
4.2.3 Conclusion and discussions . . . 49
4.3 Repair methods . . . 53
4.3.1 Experimental setup . . . 55
4.3.2 Experimental results . . . 56
4.3.2.1 Offline error analysis . . . 56
4.3.2.2 Analysis of success rate and required number of iterations for repairing solutions . . . 58
4.3.3 Conclusions and discussions . . . 60

5 Benchmarks in Dynamic Constrained Optimisation 63
5.1 Introduction . . . 63
5.2 Dynamic changes framework . . . 65
5.2.1 Constraint setup . . . 65
5.2.2 Frequency setup . . . 67
5.3 Experimental setup . . . 68
5.3.1 Ranking mechanism . . . 68
5.4 Experimental results . . . 69

5.4.1 Illustration of results for sphere . . . 69
5.4.2 Single constraint . . . 71
5.4.3 Multiple constraints . . . 74
5.5 Conclusion and future work . . . 74

6 Diversity Mechanisms in Dynamic Constrained Optimisation 77
6.1 Introduction . . . 77
6.2 Experimental setup . . . 78
6.3 Results and discussion . . . 79
6.3.1 Diversity results . . . 79
6.3.2 Statistical results . . . 80
6.3.3 Discussions . . . 84
6.4 Conclusions and future works . . . 86

7 Neural Networks in Evolutionary Dynamic Optimisation 87
7.1 Introduction . . . 87
7.2 Experimental setup . . . 88
7.2.1 Designed experiments . . . 89
7.2.2 Test problems and parameters settings . . . 90
7.3 Experimental results . . . 91
7.3.1 Frequency changes . . . 91
7.3.2 Building train data set . . . 95
7.3.3 Number and mechanism to insert predictions . . . 96
7.4 Conclusions and future works . . . 97

8 Neural Networks and Diversifying Differential Evolution 99
8.1 Introduction . . . 99
8.2 Experimental methodology . . . 101
8.3 Cross comparison of approaches . . . 103
8.4 Detailed examination of the use of neural networks . . . 109
8.4.1 Crowding . . . 109
8.4.2 Random immigrants and restart population . . . 111
8.4.3 Hyper-mutation . . . 113
8.5 Conclusions and discussions . . . 114

9 Conclusions 115

Bibliography 119

List of Figures

3.1 Building samples for neural network 33
3.2 Structure of neural network 34

5.1 Sample settings for large, medium and small changes on b_values 67
5.2 Sphere objective function and sum of constraint violation . . . 70

6.1 Diversity score (coefficient of variation of population) vs Generations 81

7.1 PCA plot of best_known positions 90
7.2 Fitness values of Rastrigin for $\tau = 1$ 91
7.3 Distribution of MOF values color-coded with τ 92
7.4 MOF-norm values considering all frequencies 93
7.5 Kruskal-Wallis test on MOF values 93
7.6 Distribution of absolute recovery rate (ARR) 94
7.7 Distribution of success rate (SR) for 30 runs 95
7.8 k-best individual selection for building NN samples 96
7.9 Number of replaced individuals 96

8.1 PCA plot of best_known positions 101
8.2 Distribution of MOF values for 20 runs 102
8.3 Heatmap of methods' rank over MOF values 104
8.4 MOF-norm values considering $\tau = 10$ 105
8.5 Kruskal-Wallis statistical test on MOF values for τ=10 106
8.6 Best error before change values over time, $\tau = 10$. 108
8.7 Euclidean distance between the best of first generation and the optimum position 109
8.8 Error of NN plus PCA for τ=10 110
8.9 Heatmap of methods' rank over absolute recovery rate (ARR) 111
8.10 Heatmap of mean values (20 runs) for success rate (SR) 111

List of Tables

2.1 Main features of the test problems 20
2.2 Designed test problems for neural networks experiments . . . 22

4.1 Average and standard deviation of modified offline error values 46
4.2 Statistical tests on the offline error values 47
4.3 Average and standard deviation of average evaluations, convergence score, progress ratio, feasibility ratio, successful ratio 51
4.4 Average and standard deviation of average evaluations, convergence score, progress ratio, feasibility ratio, successful ratio 52
4.5 Average and standard deviation of offline error values 53
4.6 Statistical tests on the offline error values 54
4.7 Average and standard deviation of: i)Success rate, ii) required number of iterations 58
4.8 Main features of each repair method 59

5.1 Testing benchmark for single constraint setup 71
5.2 Statistical test results for single constraint setup 72
5.3 Statistical test results for multiple constraint setup 73

6.1 Average and standard deviation of *MOF* values over 30 runs . 83
6.2 The 95%-confidence Kruskal-Wallis (KW) test and the Bonferroni post-hoc test on the *MOF* values 86

7.1 NN-time; time spent for NN vs overall optimisation time . . . 95

8.1 Pairwise comparison of methods on MOF values 103
8.2 NN-time; % time spent for NN vs overall optimisation time . 112

List of Abbreviations

EA	**E**volutionary **A**lgorithm
DE	**D**ifferential **E**volution
DCOP	**D**ynamic **C**onstrained **O**ptimisation **P**roblem
NN	**N**eural **N**etwork
GA	**G**enetic **A**lgorithm
PSO	**P**article **S**warm **O**ptimisation
VLS	**V**ariable **L**ocal **S**earch
CLS	**C**aos **L**ocal **S**earch
MOF	**M**odified **O**ffline **E**rror
SR	**S**uccess **R**ate
ARR	**A**bsolute **R**ecovery **R**ate
NFE	**N**umber of **F**itness **E**valuation
BEBC	**B**est **E**rror **B**efore **C**hange
VTR	**V**alue **T**o **R**each
PBIL	**P**opulation **B**ased **I**ncremental **L**earning

To my beloved parents...

Chapter 1

Introduction

To tackle complex computational problems, researchers have been looking into nature for years, both as a model and for inspiration. Across millions of years, every species has had to adapt its physical structure to its environments; consequently, optimisation occurs across many natural processes. An observation of the relationship between optimisation and biological evolution led to the development of an important paradigm of computational intelligence: evolutionary computing techniques, which perform highly complex searches and optimisation tasks [32]. Evolutionary algorithms (EAs) are most often considered to be very general search heuristics. Their main advantages compared to problem-specific search methods are their robustness, flexibility, and extensibility, and the fact that almost no domain knowledge is required for their implementation [61]. EAs are based on evolutionary operators that model problem-specific processes in natural evolution, of which the most important are crossover, mutation and selection. The basis of most evolutionary methods is a set of candidate solutions. **Crossover** combines the most promising characteristics of two or more solutions. **Mutation** adds random changes, while carefully balancing exploration and exploitation. **Selection** chooses the most promising candidate solutions in an iterative fashion, alternately with recombination and mutation. EAs have become strong optimisation algorithms for difficult continuous optimisation problems. This is despite the fact that their original versions lacked a mechanism for dealing with the constraints and dynamics of optimisation problems.

In recent studies, the presence of constraints and dynamic environments have been pointed out alongside many other sources of difficulty that exist in real-world optimisation problems [80]. These sources of difficulty include huge search spaces, noise in the objective function and the complexity of the modelling process.

Problems can be recognized as dynamic due to factors such as variation in the demand market, unpredicted events, variable resources, or estimated parameters that may change over time [16]. Likewise, they can be characterised as constrained due to restrictions in available resources, supply-demand balances, limited production capacity and maximum charging and discharging of batteries. For instance, hydro-thermal power scheduling problems [34] are dynamic since the available resources (or demand) vary over time. These problems are also constrained due to the power supply-demand balance and network constraints. Another example is parameter estimation [96] in which the parameters must be tuned dynamically. In this example data compatibility is necessary, as well as simultaneous estimation of quality and productivity parameters in real time. In addition, this problem is constrained due to the essential isothermal operation conditions, as well as mass and energy balances. All these examples (in which the objective function or/and the constraints are subject to changes) are called dynamic constrained optimisation problems (DCOPs) [67].

Although all of these problems are intrinsically dynamic, the way in which they are solved defines whether they can be considered as dynamic optimization cases. In dynamic optimisation cases, the problem can be solved using static optimisation techniques (and so is no longer of our interest) under any of the following conditions: (a) if future changes can be completely integrated into a static objective function, (b) if a single robust-to-changes solution can be provided, or (c) if only the current static instance of the time-dependent problem is taken into account. In other words, the approach in this research is not to apply an independent optimisation method to solve each problem instance separately, but to solve problems in a dynamic manner, with the algorithm detecting and responding to the changes on-the-fly.

Among the many EAs, we have selected differential evolution (DE) for our empirical studies. This is because previous competitions organized by the IEEE congress on evolutionary computation have shown DE to be one of the best choices for optimisation in continuous spaces [63, 66, 115]. It has also demonstrated competency in dynamic and constrained problems [2, 95]. Last but not least, its simple procedure helps us to understand the basic analysis of different kinds, whether this analysis involves dynamic or constraint handling techniques, or whether it entails evaluating our benchmark generator.

Throughout this research, we mainly investigate four major issues relating to these dynamic and constrained problems. In the first section, we modify common constraint handling techniques from the static domain to suit dynamic environments. We will investigate strengths and weaknesses of each

method based on the environmental change characteristics. The proper selection of constraint handling techniques is more challenging in the context of dynamic problems, since there is a mutual reinforcement relationship between constraint handling techniques and applied dynamic handling techniques. This is because, in the context of dynamic problems, the tendency of the applied method is to diversify the population through the whole search space looking for the upcoming changes. However, constraint handling techniques tend to avoid infeasible areas of the search space, directing the search toward feasible areas. A thorough study will clarify the effect of each opposing forces: constraint and dynamic handling techniques. In the second section, we propose a framework to create benchmarks. We do this because, in the first section, we identified a shortage of test cases to evaluate algorithms in dynamic constrained class of optimisation for continuous spaces. Our proposed framework can produce multiple benchmarks that can be applied to the testing of any function, and with any number of changes and dimensions of optimisation problems. In addition, we propose a ranking procedure to quantify the algorithms comparison (identifies which algorithm has better performance) without needing to know the optimal solution relating to each change. In the third section, we conduct an exhaustive empirical study of diversity mechanisms applied to solve DCOPs. This study clarifies the deficiencies and potential of each method based on the characteristics of the used test cases: disconnected feasible areas, moving optimum, etc. In the final section of the thesis, we investigate the integration of a neural network into the evolution process of DE algorithm and analyze its effectiveness in comparison to the popular diversity mechanisms like hypermutation or random immigrants. We also discuss the possibility of integrating a version of DE algorithm that incorporates neural network with diversity mechanisms to improve the results. The applied neural network is responsible for the dynamic handling portion of the algorithm, in which it predicts the future optimum position, helping DE to react to the changes properly.

Overall, this thesis is structured into the following chapters. Chapter 1 is an introduction to the thesis. It provides a background on evolutionary dynamic optimisation, dynamic constrained optimisation problems, as well as the motivation behind the research and its main concerns. In Chapter 2 presents background knowledge on dynamic optimisation, setting the basis for later discussion. Characteristics and features of dynamic problems are briefly discussed. In addition, the theory behind the problem statement is presented, as well as its formulation. Next, state-of-the art algorithms applied to solve DCOPs are reviewed, followed by benchmarks and performance

measures. In Chapter 3, we introduce our baseline algorithm and explain constraint and dynamic handling mechanisms that are used in the empirical studies of the subsequent chapters. In Chapter 4, two empirical studies are presented. First, we conduct a survey in which we compare common constraint handling mechanisms like penalty, feasibility rules, etc. We then specifically elaborate in repair methods as a promising solution to handle DCOPs. Commonly proposed repair methods will be applied and compared on the basis of a common benchmark that captures different types of environmental changes. In Chapter 5, we introduce a framework to create dynamic environments. We use this framework to design an empirical study in which we assess the ability of different algorithms to solve DCOPs. In Chapter 6, we present an exhaustive comparison over diversity mechanisms. We show that diversity mechanism are common, simple and yet effective methods of handling dynamic environments and discuss their differences in terms of solving the applied benchmark problem. In Chapter 7, we introduce the neural network as a dynamic handling mechanism to be used together with DE to tackle DCOPs. In this chapter, the neural network parameters are calibrated using some experiments. In Chapter 8, the algorithm using neural network is compared with the algorithms using common diversity mechanisms through an empirical study.

Publications outcome of this thesis

- Maryam Hasani-Shoreh, Renato Hermoza Aragonés, and Frank Neumann. "Neural Networks in Evolutionary Dynamic Constrained Optimisation: Computational Cost and Benefits." European Conference on Artificial Intelligence (ECAI-2020).
- Maryam Hasani-Shoreh, Renato Hermoza Aragonés, and Frank Neumann. "Using Neural Networks and Diversifying Differential Evolution for Dynamic Constrained Optimisation." IEEE Symposium Series on Computational Intelligence (SSCI-2020).
- Maryam Hasani-Shoreh, and Frank Neumann. "On the Use of Diversity Mechanisms in Dynamic Constrained Continuous Optimisation." International Conference on Neural Information Processing (ICONIP-2019).
- Maryam Hasani-Shoreh, Maria-Yaneli Ameca-Alducin, Wilson Blaikie, Frank Neumann and Marc Schoenauer. "On the Behaviour of Differential Evolution for Problems with Dynamic Linear Constraints." In 2019 IEEE Congress on Evolutionary Computation (CEC) (pp. 3045-3052). IEEE.

- Maria-Yaneli Ameca-Alducin, Maryam Hasani-Shoreh, and Frank Neumann. "On the Use of Repair Methods in Differential Evolution for Dynamic Constrained Optimisation." International Conference on the Applications of Evolutionary Computation. Springer, Cham, 2018.2
- Maria-Yaneli Ameca-Alducin, Maryam Hasani-Shoreh, Wilson Blaikie, Frank Neumann, Efrén Mezura-Montes. "A Comparison of Constraint Handling Techniques for Dynamic Constrained Optimisation Problems." In 2018 IEEE Congress on Evolutionary Computation (CEC) (pp. 1-8). IEEE.

Chapter 2

Dynamic Constrained Continuous Optimisation Problems

2.1 Introduction

Dynamic behaviour occurs in many real-world problems, and originates from factors such as variation in the demand market, unpredicted events, variable resources, and estimated parameters that may change over time [16, 67]. These problems, in which the objective function and/ or the constraints change over time, are called dynamic constrained optimisation problems (DCOPs) [67]. The goal in these problems is to find the optimum in each instance of the dynamic problem, given a limited computational budget. Indeed, in real-world applications, it might be the case that problems change very fast or only allow limited time for the algorithm to react. For example, in high-frequency trading, the system needs to make continual, efficient decisions in order to deal with rapid changes in financial asset pricing [42]. Also, an automatic driving system needs to rapidly adjust its operation to deal with the changing road conditions. In another scenario, a dynamic load-balancing algorithm must complete task assignments within very short time limits [139]. To solve these dynamic problems, one approach is to apply an independent optimisation method to solve each problem instance separately. However, a more efficient approach solves them through an ongoing search, in which the algorithm detects and responds to changes dynamically [86]. Our focus is the second approach in which we equip standard EAs with extra mechanisms, such as change detection and change reaction mechanisms to handle the dynamic problems. In this chapter, we introduce the main characteristics, the mathematical definition, the literature review, the benchmarks and the performance measures pertaining to these problems.

2.2 Specifications of the dynamic environments

Not any time-dependent problem is of our interest in this thesis. Our assumption is similar to the one stated by Ghosh et al. in [44]. Ghosh et al. [44] states that the fitness landscapes before and after a change should illustrate some exploitable similarities. If the whole problem changes, without any reference to the history, it is suggested to be considered as a sequence of independent problems to be solved from scratch [44]. Ghosh et al. categorise dynamic environments based on the following criteria [44], for each we may need different approaches and optimisation algorithms to tackle the problem.

1. **Frequency of change**: it defines how often the environment changes. The fitness evaluations in EAs often are the major time-determining factor. Due to this, the average number of evaluations between changes is often considered as an appropriate measure for algorithm comparisons. However, if there are other factors for computation time than evaluation, then the clock time between changes can be used for comparisons among approaches, such as in the experiments where we apply neural networks (Chapter 7 and 8).

2. **Severity of change**: it defines the strength of the changes, whether it is a slight change or a totally new condition emerges. Measures can be the genotypic distance of the optimum in two consecutive environments. Other implicit measures can be whether the new optimum may be found from the old one by a simple hill climbing approach, or the probability that the new optimum can be retrieved from the old one using a single mutation. However, more complex measures can be defined such as the correlation of old to new fitness values of all points in the search space.

3. **Predictability of change**: it defines whether there is a pattern or trend in the changes that can be predicted. This pattern can be in any aspect such as direction, time, or severity of the next change.

4. **Cycle length/ cycle accuracy**: it defines whether the optimum return to its exact or close previous locations. A cycle can be measured as the average number of environmental states between two consecutive encounters of the same (or a very similar) state. If the new state is not precisely the same but a slight variant, then the distance of the new to the previously encountered solution is important.

5. **Visibility of change**: it defines whether the changes are explicitly known to the system or they need to be detected by the algorithm through separate mechanisms.

6. **Necessity to change representation**: it determines whether the genetic representation is impacted with a change, in case for instance when the problem dimension has changed.

7. **Aspect of change**: it defines whether the change happened in the objective function, the problem instance, or the constraints.

2.3 Problem statement

A dynamic constrained optimisation problem is mathematically defined as follows: Find $\vec{x}$, at each time t, which:

$$\min_{\vec{x} \in F_t \subseteq [L,U]} f(\vec{x}, t) \tag{2.1}$$

where $t \in N^+$ is the current time,

$$\begin{aligned} &[L, U] = \{\vec{x} = (x_1, x_2, ..., x_D) \mid L_i \leq x_i \leq U_i, \\ &i = 1 \dots D\} \end{aligned} \tag{2.2}$$

is the search space, subject to:

$$\begin{aligned} &F_t = \{\vec{x} \mid \vec{x} \in [L, U], g_i(\vec{x}, t) \leq 0, i = 1, \dots, m, \\ &h_j(\vec{x}, t) = 0, j = 1, \dots, p\} \end{aligned} \tag{2.3}$$

is called a 'feasible region' at time t.

$\forall \vec{x} \in F_t$ if there exists a solution $\vec{x}^* \in F_t$ such that $f(\vec{x}^*, t) \leq f(\vec{x}, t)$, then $\vec{x}^*$ is called a 'feasible optimum solution' and $f(\vec{x}^*, t)$ is called the 'feasible optimum value' at time t.

Variable t (time), refers to the number of change steps that have occurred up until a set point in time in the problem. In this thesis, two different methods are used to impose the changes. First, we use one that is more common in the DCOPs literature, that is, a method in which a change occurs after a specific number of fitness evaluations. In this case, time is a discrete variable. In the second approach, we use wall clock timing; a change happens after the actual running time of the algorithm. We adopt the second approach for the experiments with neural networks.

2.4 Evolutionary algorithms for dynamic problems: state of the art

As mentioned in the introduction, in a dynamic problem, solving each problem instance separately from scratch without reusing information from the past is inefficient, particularly in cases when the solution to the new problem might not differ too much from the solution in the old problem. Thus, it would be more practically efficient to employ an optimisation algorithm that can continuously adapt the solution to a changing environment, reusing the previous information. Since EAs have much in common with natural evolution, and since, in nature, adaptation is a continuous and continuing process, EAs seem to be a suitable candidate for dealing with dynamic problems [86]. The main drawback in using standard EAs for dynamic optimisation problems appears to be that EAs eventually converge to an optimum. As a result, they lose their diversity which is required to explore the search space. Consequently, they also lose their ability to adapt to a change in the environment when such a change occurs. In recent years, researchers have addressed this issue in various ways. This section provides a survey of the state of the art in the field, to allow a closer look at recent approaches, and to serve as a basis for future research.

2.4.1 Diversity introducing

In dynamic optimisation, premature convergence is not ideal, as change in one area of the dynamic landscape (if no member of the algorithm exists in this area) will prevent the algorithm from reacting to the change. As such, the algorithm may not be able to track the optimum. One simple, intuitive approach to prevent premature convergence would be to increase the diversity of an EA after a change has been detected.

First methods following this approach are hyper-mutation [24] and variable local search (VLS) [120]. In an adaptive mutation, an operator called hyper-mutation is introduced. The mutation rate of this operator is a multiplication of the normal mutation rate and a hyper-mutation factor [24]. The hyper-mutation is activated only if a change is detected. In the VLS algorithm, the mutation size is controlled by a variable local search range. This range is determined by a formula that is adjustable during the search [121], or can be adapted using a learning strategy stemming from the feature partitioning algorithm [120]. In [91], hyper-mutation is used in an EA in which detectors are placed near the boundary of feasible regions. When the feasibility of these

detectors changes, the EA increases its mutation rate. This raises the diversity level of the population to track the moving feasible regions and decreases again once the change has been tracked successfully. In [103] an adaptive genetic programming is proposed. As well as increasing the mutation rate, this method reduces elitism and increases crossover probability after a change.

For particle swarm optimisation (PSO), this approach is introduced using a simple mechanism in which a part of the swarm or the whole swarm is re-diversified using randomisation after a change is detected [52]. In [56], in addition to employing partial re-diversification, this strategy divides the swarm into several sub-swarms after each change. This process is performed for a certain number of generations, which serves to prevent the swarm from converging on the old position of the global optimum too quickly. In [30] a cultural-based PSO is proposed, where after a change, the swarms are re-diversified using a framework of knowledge inspired by the belief space in cultural algorithms. In [124], a new adaptive method is proposed in which, after a change, individuals are relocated to a position within a specific radius using mutation. This position is estimated based on the individual's performance history. The more sensitive the individual is to changes, the larger the radius.

Methods following this approach have the advantage of focusing fully on the search process and only reacting to changes once they are detected, since they do not need to constantly maintain diversity [86]. In addition, methods such as hyper-mutation have good results when solving problems with highly frequent changes (where these changes are small or medium). This is because distributing individuals around an optimum resembles a type of "local search", which is a useful way to observe the nearby places of this optimum. One of the shortcomings of these methods is that they are dependent on whether changes are known or easily detectable. If the changes are not properly distinguished, these methods fail to provoke. In addition, they exhibit difficulty in identifying the correct mutation size or the number of sub-swarms [86].

2.4.2 Maintaining diversity

The aim of this approach is to keep population diversity throughout the search process. This helps to avoid the possibility that the whole population will converge on one place, making it unable to track the changing optimum. Methods following this approach usually do not detect changes explicitly; instead, they rely on their diversity to cope adaptively with changes. Examples of this

approach are the random immigrants method [46], fitness sharing [8], thermodynamical GA [82], sentinel placement [83], population-based incremental learning [132], several PSO variants [10, 11, 55], and dynamic evolutionary multi-objective optimisation [1, 21, 117]. In the random immigrants method, a number of randomly generated individuals are added to the population at every generation in order to maintain diversity. In [83], a sentinel placement method is proposed that is slightly different compared to the random immigrants method. Within the sentinel placement method, a number of sentinels are initialised and specifically distributed throughout the search space. Experiments show that this method might achieve better results than the random immigrants method and hyper-mutation in problems with large and chaotic changes [83]. On the basis of the population-based incremental learning (PBIL) algorithm, which is a simple combination of population-based EA and incremental learning, two approaches (namely parallel PBIL and dual PBIL) were proposed in [132]. PBIL has an adjustable probability vector which is used to generate individuals. After each generation, the probability vector is updated based on the best found solutions. In this method, it is ensured that the vector will gradually learn the appropriate value in order to create high quality individuals. In parallel PBIL, the results are improved by maintaining two parallel probability vectors: a vector similar to the original one in PBIL, and a random initialised probability dedicated to maintaining diversity during the search. The two vectors are sampled and updated independently so that their sample sizes can be adjusted based on their relative performance. To increase the ability of parallel PBIL to deal with large changes, dual PBIL is proposed where two probability vectors are dual to each other [132]. During the search, only one of the vectors needs to learn from the best generated solution, because the other one will change automatically with the first one.

The other way to maintain diversity is by rewarding individuals that are genetically different to their parents [122]. In this approach, besides to a standard GA population, the algorithm maintains two other populations: in the first one individuals are chosen based on their Hamming distance to their parents and in the second one the individuals are selected based on their fitness improvement compared to their parents. In evolutionary strategy, diversity can also be maintained by preventing the strategy parameters from converging to 0 [60]. For PSO [10, 11], a repulsion mechanism (inspired by the atom field) is applied to prohibit particles from becoming too close to each other. In this mechanism, each swarm is comprised of a nucleus and a cloud of charged particles which are responsible for maintaining diversity. In [30], both the particle selection and replacement mechanisms are modified so that the

most diversified particles (based on the Hamming distance) are selected and particles that have similar positions are replaced. In the compound PSO [68], the degree to which particles deviate from their original directions becomes larger when the velocities become smaller, and distance information is used to decide for selecting a particle.

In [21], it is proposed that multiple objectives be used to maintain diversity. The dynamic problem is represented as a two-objective problem. The first objective is the original objective, and the second is a special objective created to maintain diversity. Other examples of using multiple objectives to maintain diversity can be found in [1, 117]. In the latter, six different types of objective are proposed, including retaining more old solutions; retaining more random solutions; reversing the first objective; keeping a distance from the closest neighbour; keeping a distance from all individuals; and keeping a distance from the best individual.

Methods following this approach are suitable for solving problems with severe changes, and, in certain situations, solving problems with large changes [86]. For example, in [85, 91], it has been shown that the random-immigrant method helps significantly improve performance in dynamic constrained problems where changes are severe due to the presence of disconnected feasible regions. In addition, these approaches show competitive results for solving problems with rare changes [132], an algorithm with high diversity may have enough time to converge in the case of such problems. They are also effective in solving problems with competing peaks [22]. However, methods that maintain diversity throughout the search also have some disadvantages. For instance, they are slow, as continuously focusing on diversity may slow the algorithm down, or even distract the optimisation process. They are often deficient when the changes are small, since most methods following this approach maintain their diversity by adding some stochastic element throughout the search. Obviously, situations in which the optimum just take a slight move away from their previous positions will make the algorithm less effective in dealing with small changes [86].

2.4.3 Memory-based

In cases, when there are repeated occurrences of situations (periodically changing environments), supplying the EA with some sort of memory might allow it to store high quality solutions and reuse them later. Memory may be provided in two general ways: implicitly, by using redundant representations,

or explicitly, by introducing an extra memory and then using strategies to deposit and retrieve solutions from it.

Redundant coding using diploid genomes is the most common implicit memory used in EAs for solving dynamic problems [86, 130]. A diploid EA is usually an algorithm whose chromosomes contain two alleles at each locus. In multiploid approaches for dynamic environments, the following three components need to be incorporated: representation of the redundant code; readjustment of the dominance of alleles; and checking for changes. One typical way to represent the dominance of alleles is to use a table [84] or a mask [27] to map between genotypes and phenotypes. The dominance relationships among alleles can then be changed adaptively depending on the detection of changes in the landscape.

Conversely, methods following an explicit approach need to decide on the content of the explicit memory, which can be either direct or associative. In most cases, direct memories are the previous good solutions/local optimum [14, 30, 125, 127, 129, 134]. In [134], for certain circumstances the most diversified solutions are also selected for the memory. In [30], a set of previous positions and the corresponding fitness values of each individual may also be stored in the memory. Various types of information can be included in the associative memory, including information about the environment at the considered time [39]; the list of environmental states and state transition probabilities [110]; the probability vector that created the best solutions [125]; distribution statistics information for the population at the considered time [127]; the probability of the occurrence of good solutions in each area of the landscape [101, 102]; or the probability of likely feasible regions [100].

In methods using explicit memory, it is important to know how to update the memory. Generally, the best found elements (direct or associative) of the current generation will be used to update the memory. These newly found elements will replace some existing elements in the memory, which can include the oldest element in the memory [39, 111, 123], the element with the least contribution to the diversity of the population [14, 39, 111, 125, 138], or the element with the least contribution to fitness [39]. The other important criterion in memory approaches is the understanding of when to update the memory. Ideally if we know when a change happens, then the most suitable time to update the memory is immediately after the change happens. Otherwise, the memory may also be updated after each or a certain number of generations. Usually the best elements in the memory will be used to replace the worst individuals in the population.

Some works have shown that redundant coding does not ensure sufficient diversity for population to adapt to random changes [14, 15]. To improve their results, several studies have tried to combine memory-based approaches with diversity schemes [111, 128]. Redundant coding approaches might not be promising for cases where the number of oscillating states is large, since the redundant code might also become too large, and hence reduce the performance of the algorithm. Furthermore, it might not always be possible in practice to know the number of oscillating states of the problem. Without this information, it is impossible to design an appropriate representation for the redundant code. The information stored in the memory might become redundant (and obsolete) when the environment changes.

2.4.4 Prediction-based

In certain cases, changes in dynamic environments may exhibit some patterns that are predictable. Where this occurs, it might be effective to attempt to learn these type of patterns from the previous search experience and, based on these patterns, to aim at predicting changes in the future. Based on this idea, some studies have attempted to exploit the predictability of dynamic environments. Obviously, memory approaches, which are proposed to deal with periodical changes, can also be considered a special type of prediction approach. However, methods following the prediction approach are often capable of using their memory to deal with greater variety of change types than cyclic/recurrent changes.

A common prediction approach is to predict the movement of the moving optima. In [51], an autoregressive forecasting technique is combined with an EA and is used to predict the location of the next optimum solution after a change is detected. The forecasting model (time series model) is created using a sequence of optimum positions found in the past. A similar approach has been proposed in [106], where the movement of optima was predicted using Kalman filters. The predicted information is incorporated into an EA in three ways. First, modifying the mutation operator by introducing some bias to favour the search toward the predicted region. Second, individuals close to the predicted position are rewarded by modifying the fitness function. Third, some individuals (called as gifts) are generated at the predicted position, and introduced into the population.

Another approach is to predict the locations to which individuals should be re-initialised when a change occurs. In [137], this approach is used to solve two dynamic multi-objective optimisation benchmark problems in two ways.

First, the solutions in the Pareto set from the previous change periods were used as a time series to predict the next re-initialization locations. Second, to improve the chance of the initial population to cover the new Pareto set, the predicted re-initialization population is perturbed with a Gaussian noise whose variance is estimated based on the historical data. Compared with random-initialization, the approach was able to achieve better results on the two tested problems. Another approach to estimate the areas to re-initialize individuals after a change occurs is the relocation variable method [124].

Another interesting approach is to predict the time when the next change will occur and which possible environments will appear in the next change [110, 112]. In these works, the authors used two prediction modules to predict two different factors. The first module, which uses either a linear regression [110] or a non-linear regression [112], is used to estimate the generation when the next change will occur. The second module, which uses Markov chain, monitors the transitions of previous environments and based on this data provides estimations of which environment will appear in the next change. In relation to prediction approaches, some studies have addressed time-linkage problems [1]. The authors of these studies [12, 13] have suggested that the only way to solve such problems effectively is to predict future changes and take into account the possible future outcomes when solving the problems online.

Besides these methods, neural networks (NNs) have gained increasing attention in recent years [58, 70, 72, 73]. In [73], a temporal convolutional network with Monte Carlo dropout is used to predict the next optimum position. The authors propose to control the influence of the prediction via estimation of the prediction uncertainty. In [72] a recurrent NN is proposed that is best suited for objective functions where the optimum movement follows a recurrent pattern. In other works [58, 70], where the change pattern is not stable, it is proposed to directly construct a transfer model of the solutions/fitness using NNs, considering the correlation and difference between the two consecutive environments.

Generally, methods following prediction approach may become very effective if their predictions are correct. In this case, the algorithms can detect, track, or find the global optima quickly, as shown in [51, 109, 126]. However, prediction-based algorithms are prone to training errors. These errors might occur because of wrong training data. For example, if the algorithm has not performed successfully in the previous change periods, the history data collected by the algorithm might not be helpful for the prediction or might even

[1] problems where the current solutions made by the algorithms can influence the future dynamics

provide the wrong training data. Situations featuring lack of training data can be characterised as follows: as in the case of any learning, predicting, and forecasting model, the algorithms may need a large enough set of training data to produce satisfying results. In addition, prediction can only be started after sufficient training data has been collected [12, 13, 110, 112]. In the case of dynamic optimisation, where there is a need to find and track the optima as quickly as possible, this might present a disadvantage.

Overall, the nature of the dynamic problems plays a role in the success of these approaches [86]. If changes in the dynamic environment are easily predictable (trends of linear, periodic or deterministic is to be found), the result is expected to be promising, as can be seen in [51, 106]. However, if the changes are stochastic, or history data is misleading, prediction approaches might not lead to satisfactory results.

2.4.5 Multi-population-based

In this approach, multiple sub-populations are used simultaneously, each handling a separate area of the search space and becoming responsible for a different task. Methods following this approach mainly have two goals. First, assigning different tasks to each sub-population, for example P_{search} to search and P_{track} to track. Second, dividing the sub-populations in a way to have the best diversity and ensure the sub-populations are not overlapped [86].

For the first goal, different approaches has been proposed. One approach was proposed in [93], called as shifting balance GA. In this method, there are a number of small populations in P_{search}, searching for new solutions, and there is only one large population in P_{track} to track changing peaks. Conversely, the self-organizing scouts method [17], uses the main large population to search for optimum, and dedicates several small populations to track any change of each optimum. Once the main population finds a new peak, it will create a new sub-population to track changes in that peak. This approach was adopted in different types of EAs and meta-heuristics, DE [71, 74], and PSO [41].

Similarly, an algorithm named RepairGA was proposed in [85, 89], that uses one large population to search and a smaller population to track changes. The difference between RepairGA and previous approaches is that its two sub-populations can overlap in the search space. The distinction between the two sub-populations is that the main population accepts both infeasible and feasible individuals, while the sub-population only includes feasible individuals. Another approach, the multinational GA, introduced in [119], implements both the features of P_{search} and P_{track} into each sub-population. Meaning each

population can both search for new solutions and track changes. Whenever a sub-population detects a new optimum, it will split into two sub-populations to assure that each sub-population only tracks one optimum at a time. This approach has also been implemented in artificial immune algorithms [33], and PSO-based algorithms. For example, speciation PSO [64] is proposed where each sub-population, or species, is a hyper-sphere defined by the best fit individual and a specific radius. The other example, the clustering PSO [62, 131], also has multi-swarms with equal roles.

Relating to the goal of assigning the tasks to sub-populations, it should be noted that in dynamic optimisation, multiple populations are used not only for the purpose of exploring different parts of the search space, but also for the purpose of co-evolution [45, 85, 89] or maintaining diversity and balancing exploitation and exploration [122].

For the second goal, dividing the sub-populations and prevent overlapping between sub-populations, clustering probably is the most common approach. In clustering, some individuals are selected as the centres, then each sub-population is defined as a hyper-cube with a specific size (all individuals within the range are part of that sub-population). SOS [17], is one of the earliest methods that adopt this approach. It keeps the sub-populations from being overlapped by using an idea coming from the forking genetic algorithm [118] which divides up the space. Whenever the main population in P_{search} finds a new optimum, it creates a new population in P_{track} and assigns this new population to the optimum. To separate the sub-populations, SOS [17] restrict each sub-population to a hyper-cube determined by a centre (the most fit individual in the population) and a pre-determined range. If an individual that belongs to one of the sub-populations enters to the area monitored by another sub-population, this individual will be discarded and re-initialized. The same forking approach is used in other EAs, such as DE [71, 74] or PSO in multi-swarm PSO [10]. Swarms are also divided into sub-swarm in the same way as in SOS, so that each swarm caters a different peak. In addition, multi-swarm PSO also maintains a similar mechanism, known as anti-convergence, to the P_{search} in SOS so that there is always one free swarm to continue exploring the search space. Another example is speciation PSO [64], where each species is a hyper-sphere whose centre is the best-fit individual in the species and each species can be used to track a peak.

For clustering approaches, not always the best solutions are nominated as the clusters' centres. In other studies [62, 123], density-based clustering methods are used to divide the sub-populations allowing the algorithms to explore different parts of the search space. Due to the pair-wise distance calculations

among particles these techniques are computationally expensive. The second approach is to incorporate some penalty or rewarding strategies to isolate the sub-populations, of which SBGA [93] is a typical example. SBGA maintains the separation of populations by selecting individuals in P_{search} for reproduction based on their distance from the core in P_{track} rather than their original fitness values. The third approach is to estimate the basins of attractions of peaks using these basins as the separate regions for each sub-population, such as MGA [119]. In this work the authors introduced a mechanism called 'hill-valley detection' that by considering two individuals, they calculate the fitness of several random samples on the line connecting these two individuals. If the fitness in a sample point is lower than that of the two individuals, then a valley is detected. If a sub-population contains more than one valley, it will be split.

Methods following the multi-population approach can maintain enough diversity for the algorithm to adaptively start a new search once a new change emerges [15]. These approaches are also successful in tracking the changes of multiple optimum, as analysed in many existing studies on multi-population [15, 119]. They can be very effective for solving problems with competing peaks or multimodal problems.

The drawbacks of the multi-population approaches are the number of populations, the search area under each population, and the size of each population are difficult to be defined. Too many sub-populations may slow down the search [10].

2.5 Benchmarks in dynamic optimisation

As well as developing algorithms, it is important to test them using a comprehensive benchmark that considers a range of characteristics. A range of benchmarks have been proposed to test the relevant algorithms for discrete spaces [105], and/or multi-objective optimisation in dynamic environments [59]. However, for continuous spaces in single objective optimisation, the most commonly used benchmark so far is that proposed in [86].

According to Nguyen benchmark [86], dynamic changes are applied by adding time-dependent terms to the objective function and the constraints of one of the functions (G_24) of the static benchmark proposed in CEC 2006 [66]. Table 2.1 summarises the main features of these test problems. This test suite comprises 22 problems in total. The first 18 test cases are from [90],

TABLE 2.1: Main features of the test problems [18, 90].

Problem	Obj. Function	Constraints	DFR	SwO	bNAO	OICB	OISB	PFR $S = 20$
G24_u	Dynamic	No Constraints	1	No	No	No	Yes	100%
G24_1	Dynamic	Static	2	Yes	No	Yes	No	44.61%
G24_f	Static	Static	2	No	No	Yes	No	44.61%
G24_uf	Static	No Constraints	1	No	No	No	Yes	100%
G24_2*	Dynamic	Static	2	Yes	No	Yes and No	Yes and No	44.61%
G24_2u	Dynamic	No Constraints	1	No	No	No	Yes	100%
G24_3	Static	Dynamic	2-3	No	Yes	Yes	No	7.29-44.61%
G24_3b	Dynamic	Dynamic	2-3	Yes	No	Yes	No	7.29-44.61%
G24_3f	Static	Static	3	No	No	Yes	No	7.29%
G24_4	Dynamic	Dynamic	2-3	Yes	No	Yes	No	7.29-44.61%
G24_5*	Dynamic	Dynamic	2-3	Yes	No	Yes and No	Yes and No	7.29-44.61%
G24_6a	Dynamic	Static	2	Yes	No	No	Yes	17%
G24_6b	Dynamic	Static	1	No	No	No	Yes	50.5%
G24_6c	Dynamic	Static	2	Yes	No	No	Yes	33.63%
G24_6d	Dynamic	Static	2	Yes	No	No	Yes	17%
G24_7	Static	Dynamic	2-3	No	No	Yes	No	7.29-44.61%
G24_8a	Dynamic	No Constraints	1	No	No	No	No	100%
G24_8b	Dynamic	Static	2	Yes	No	Yes	No	44.61%
G24v_3	Static	Dynamic	2	No	No	Yes	No	0.37%
G24v_3b	Dynamic	Dynamic	2	Yes	No	Yes	No	0.37%
G24w_3	Static	Dynamic	3	No	No	Yes	No	0.10%
G24w_3b	Dynamic	Dynamic	3	Yes	No	Yes	No	0.10%

Dynamic	The function is dynamic	Static	There is no change
DFR	Number of disconnected feasible regions	PFR	Percentage of feasible region
OICB	Global optimum is in the constraint boundary	OISB	Global optimum is in the search boundary
SwO	Switched global optimum between disconnected regions		
bNAO	Better newly appear optimum without changing existing ones		
*	In some change periods, the landscape either is a plateau or contains infinite number of optima and all optima (including the existing optimum) lie in a line parallel to one of the axes		

which captures various characteristics of DCOPs, such as their multiple disconnected feasible regions, gradually moving feasible regions and global optimum switching between different feasible regions. However, there are parameters in this test suite which are defined to alter the severity of changes in the environment, this benchmark is based on only one objective function and the transformation of this function. Thus, it is not applicable to the testing of different functions with the purpose of considering a range of characteristics such as multi-modality, convex versus non-convex, etc. Moreover, the proposed problem is two-dimensional and is not sufficiently flexible to be applied to larger problem dimensions. In addition, the feasible regions of the dynamic constraint function in this benchmark are very large, which might not be sufficiently complicated to challenge an algorithm.

Bu et al. [19], introduce one variant from Nguyen benchmark suite with a parameter to control the size and the number of the feasible regions (the last 4 test cases presented in Table 2.1). A similar benchmark is proposed in [135]. This benchmark is based on dynamic transformations introduced by Nguyen in [86, 90]. However, the problem information, including the number of feasible regions, the global optimum, and the dynamics of each feasible region, is lacking in this benchmark. The lack of such information makes it difficult to measure and analyse the performance of an algorithm, and this is probably the reason that this benchmark has become less popular than Nguyen's benchmark [86].

In terms of having a scalable and flexible benchmark, [63] propose a dynamic benchmark generator that is designed with the idea of constructing dynamic

environments across binary, real, and combinatorial solution spaces. The dynamism is obtained by tuning some system control parameters, creating six change types: 'small step', 'large step', 'random', 'chaotic', 'recurrent', and 'recurrent change with noise'.

There are shortcomings in the current benchmarks, and only a couple of benchmarks exist for the purpose of testing algorithms for dynamic problems in continuous spaces. So in this thesis (Chapter 5) we propose a framework to create benchmarks. While the aforementioned benchmark generator's main focus is on creating dynamic objective functions, our focus is on creating dynamic constraints. Our motivation comes from characteristics of some real-world problems such as the dynamic linear constraints of the power system scheduling problem (which are due to variable demand and availability of resources over-time). For a clearer insight into the effects of constraint changes, we keep the objective function static. Indeed, this is the case in some real world problems in which only constraints will change, such as the problem of hydro-thermal power scheduling in continuous spaces [36] or the problem of ship scheduling in discrete spaces [75].

Dynamic changes are imposed by the translation and rotation of the constraint's hyperplane. Some examples of these two operations on constraints in a real-world dynamic environment are the reduction and increase of demand that occur regularly in a power system (hyperplane translation), and changes to the share of each plant's power production (hyperplane rotation) [81]. Our proposed benchmark generator is flexible (frequency and severity of changes, number of environmental changes, and dimension of the problem); simple to implement (with any objective function), analyse, or evaluate; and computationally efficient. Finally our benchmark generator allows for the formation of conjectures about real-world problems. Chapter 5 discusses details of this framework, as well as presenting experimental investigations based around it.

We also extend our framework to the creation of benchmarks for the testing of algorithms in neural network experiments (Chapter 7, and 8). Dynamic environments are created in two general cases for common functions in the literature: Sphere, Rosenbrock and Rastrigin. In the first two experiments, the objective function is constant, while the constraints change, and for the third and fourth experiments, we define the problem as unconstrained, while its objective function is dynamic. Details of the dynamism which we have built into each experiment are presented in Table 2.2.

TABLE 2.2: Designed test problems for neural networks experiments

exp1	Uniformly random changes on the boundaries of one linear constraint	$b[t+1] = b[t] + \mathcal{U}(lk, uk)$
exp2	Patterned sinusoidal changes on the boundaries of one linear constraint	$b[t+1] = 5 \cdot sin(b[t]) + \mathcal{N}(0, 0.5)$
exp3	Linear transformation of the optimum position	$\vec{X_{t+1}} = \vec{X_t} + 0.1t$
exp4	Transformation of the optimum position in sinusoidal pattern with random amplitudes	$\vec{X_{t+1}} = \vec{X_t} + 5 \cdot \mathcal{N}(0, 0.5) \cdot sin(\frac{\pi}{2}t)$

In the first two experiments, the changes are targeted on b values (constraint boundaries) of one linear constraint in the form of $a_i x_i \leq b$ [50][2].

2.6 Performance measures

In this section, we gather all the performance metrics that are applied in the rest of this thesis in different experiments. The last two relates to experiments using neural networks.

Modified offline error (MOF) represents the average of the sum of errors in each generation divided by the total generations [87].

$$MOF = \frac{1}{G_{max}} \sum_{G=1}^{G_{max}} (|f(\vec{x}^*, t) - f(\vec{x}_{best,G}, t)|) \tag{2.4}$$

Where G_{max} is the maximum generation, $f(\vec{x}^*, t)$ is the global optimum at current time t, and $f(\vec{x}_{best,G}, t)$ represents the best solution found so far at generation G at current time t. Only feasible solutions are considered to calculate the best errors at every generation. If there were no feasible solution at a particular generation, the worst possible value that a feasible particle can have would be taken.

Absolute recovery rate introduced in [87] is used to analyze the convergence behaviour of the algorithms in dynamic environments. This measure infers how quick an algorithm starts converging to the global optimum before the next change occurs. An important observation about this metric is that it reports the speed of convergence relatively to the first achieved solution [72].

$$ARR = \frac{1}{T_{max}} \sum_{t=1}^{T_{max}} \left(\frac{\sum_{G=1}^{G_{max}(t)} |f_{best}(t, G) - f_{best}(t, 1)|}{G_{max}(t)[f^*(t) - f_{best}(t, 1)]}\right) \tag{2.5}$$

Success rate (SR) calculates in how many times (over all times) each algorithm is successful to reach to ϵ-precision from the optimum before reaching to the next change.

[2] a_i is the coefficient of the variables in the linear constraint

Feasibility ratio (FR_t): The feasibility ratio consists on the number of feasible solutions per time (f_t) divided by the total number of times performed (T), as indicated in Equation 2.6.

$$FR_t = f_t / T \tag{2.6}$$

The range of values for FR_t goes from 0 to 1, where 1 means that in all times feasible solutions were found. In this way, a higher value is preferred.

Success ratio (SR_t): The success ratio is calculated by the ratio of the number of successful times (s_t) [3] to the total number of times performed (T), as indicated in Equation 2.14.

$$SR_t = s_t / T \tag{2.7}$$

Similar to FR_t, the range of values for SR_t goes from 0 to 1, where 1 means that in all of the times successful solutions were found. Therefore, a higher value is preferred.

Average evaluations (AE_t): This measure is calculated by averaging the number of evaluations required on each successful run to find the first successful solution.

$$AE_t = (1/s_t) \cdot \sum_{i=1}^{s_t} (E_t) \tag{2.8}$$

where E_t is the number of evaluations required to find the first successful solution in any successful time. For E_t, a lower value is preferred because it means that the average computational cost is lower for an algorithm to reach the vicinity of the feasible optimum solution.

Convergence score (CS_t): The two previous performance measures (SR_t and AE_t) are combined to measure the speed and reliability of an algorithm through a successful performance.

$$CS_t = AE_t / SR_t \tag{2.9}$$

For this measure, a lower value is preferred because it means a better ratio between speed and consistency of the algorithm.

[3] a time is considered successful if the best solution for this time is near to the optima with a precision (10^{-4})

Progress ratio (PR_t): The objective is to measure the improvement capability of the algorithm within the feasible region of the search space. For this measure high values are preferred because they indicate a higher improvement of the first feasible solution found (see Equation 2.10).

$$PR_t = \begin{cases} \left| \ln \sqrt{\frac{f(\vec{x}_{first,G},t)}{f(\vec{x}_{best,G},t)}} \right| & \text{if } f(\vec{x}_{best,G},t) > 0 \\ \left| \ln \sqrt{\frac{f(\vec{x}_{first,G},t)+1}{f(\vec{x}_{best,G},t)+1}} \right| & \text{if } f(\vec{x}_{best,G},t) = 0 \\ \left| \ln \sqrt{\frac{f(\vec{x}_{first,G},t)+2|f(\vec{x}_{best,G},t)|}{f(\vec{x}_{best,G},t)+2|f(\vec{x}_{best,G},t)|}} \right| & \text{if } f(\vec{x}_{best,G},t) < 0 \end{cases} \tag{2.10}$$

Where $f(\vec{x}_{first,G},t)$ is the value of the objective function of the first feasible solution found and $f(\vec{x}_{best,G},t)$ is the value of the objective function of the best solution found. For this measure, statistical values are also provided.

Success rate: This measure is calculated such that considers how many of the infeasible solutions were successful to be repaired after 100 iterations. For each infeasible solution, a repair is needed and at the end of repair iteration(Maximum 100 tries), if the solution is feasible a counter is increased. In another words, it is considered a success if before achieving to the maximum number of allowed iterations for repair (100 in our case) a solution is feasible. The total number of these successful repaired solutions (s) divided by the total number of solutions that need repair (n_T)is equal to success rate percentage. Based on this, the repair methods with success rate values equals to 100%, are able to convert all the solutions.

$$s_r = \frac{s}{n_T} \tag{2.11}$$

Required number of iterations: In order to distinguish the difference between the number of evaluations that each method consumes for repairing the solution, a measurement is defined called as required number of iterations (rn_i). In this way, it is possible to compare the efficiency of each repair method. The range of values of this measure is $\in [1-100]$. The more efficient method uses lower number of evaluations in order to repair an infeasible solution. The final amount for this measurement value is the average between the number of tries taken to convert each infeasible solution into feasible one.

Best error before change (BEBC) is another common measure that considers the behaviour of algorithm only in the last solution achieved before next

change happens.

$$BEBC = \frac{1}{T_{max}} \sum_{t=1}^{T_{max}} (|f(\vec{x}^*, t) - f(\vec{x}_{best}, t)|) \tag{2.12}$$

Number of fitness evaluations (NFE**)** needed at each time to reach to an ϵ-precision from the global optimum are averaged over all the times for this measure. The termination criteria is to find a value smaller than the ϵ-level from the global optimum (value to reach (VTR)) before reaching to the next change.

$$\begin{aligned} NFE &= \frac{1}{t_{end} - t_0} \sum_{t=t_0}^{t_{end}} NFE_t \\ VTR &= \frac{|f(\vec{x}^*, t) - f(\vec{x}_{best,G}, t)|}{|f(\vec{x}^*, t)|} \end{aligned} \tag{2.13}$$

Success rate (SR**)** calculates the percentage of the number of times each algorithm is successful to reach to ϵ-precision from the global optimum (VTR) over all time scale.

$$SR = \frac{\text{number of times reached to VTR}}{t_{end} - t_0} \tag{2.14}$$

Diversity: Diversity measures differences among individuals at distinct levels; genotypic: considers individuals position within the search space or phenotypic: evaluate populations fitness distribution. We choose a genotypic measure as it is more common in the literature. For this purpose, we measure relative standard deviation of the population (known as coefficient of variation): $CV = \frac{\sigma}{\mu}$ at each generation, where σ is the standard deviation and μ is the mean of the population.

NN-Error reports the Euclidean distance of the predicted solution and the best_known. Lower values are preferred.

NN-time reports the percentage of the time spent to train and use NN per overall optimisation time.

Chapter 3

Designing Evolutionary Algorithms for Dynamic Constrained Continuous Problems

3.1 Introduction

Optimisation is intrinsically found in many natural processes, considering in nature every species had to adapt their physical structures to fit to their environments. This observation inspired the researchers to develop an important class of computational intelligence, the evolutionary computing techniques to perform very complex searches [32]. Evolutionary computation is based on iterative progress, examples of which include growth or development in a population. This population is then selected in a guided random search using parallel processing to acquire the desired objective. Originally, research on evolutionary computation was focused on optimisation of static, non-changing problems. Many real-world optimisation problems, however, are dynamic. On this basis, optimisation methods need to be capable of continuously adapting the solution to a changing environment. If the optimisation problem is dynamic, the goal is no longer to find the optimum, but to track their progression through the search space using some extra mechanisms. In this chapter, we first introduce differential evolution as our chosen algorithm for the experimental studies in this thesis. We then introduce the mechanisms we need to include in the baseline DE, namely change detection, change reaction and constrained handling techniques. All of these techniques will allow the algorithm to solve dynamic and constrained problems.

Algorithm 1 Dynamic differential evolution (DDE)

Create and evaluate a randomly initial population $\vec{x}_{i,G}, i = 1, \ldots, NP$
for $G \leftarrow 1$ to G_{max} **do**
  for $i \leftarrow 1$ to NP **do**
    Change detection mechanism $(\vec{x}_{i,G})$
    Randomly select $r0 \neq r1 \neq r2 \neq i$
    $J_{rand} = randint[1, D]$
    for $j \leftarrow 1$ to D **do**
      if $rand_j \leq CR$ Or $j = J_{rand}$ **then**
        $u_{i,j,G} = x_{r1,j,G} + F(x_{r2,j,G} - x_{r3,j,G})$
      else
        $u_{i,j,G} = x_{i,j,G}$
      end if
    end for
    Select $u_{i,j,G}$ or $x_{i,j,G}$ based on the constraint handling
  end for
end for

3.2 Differential evolution

Differential evolution (DE) is known as one of the most competitive, reliable and versatile evolutionary algorithms for the optimisation of continuous spaces [31]. Recent studies show that, despite its simplicity, DE exhibits much better performance than several other algorithms of current interest on a variety of problems including multi-objective [98], multi-modal [9], large-scale [92], expensive [40], constrained [20] and dynamic optimisation problems [2, 95]. It exhibits remarkable performance in terms of final accuracy, computational speed, and robustness. The space complexity of DE is low compared to that of some of the most competitive real parameter optimisers like CMA-ES [47]. This means that DE is capable of handling large-scale and expensive optimisation problems. Indeed, the variants of DE have been selected as top ranks among other EAs in various competitions organized by IEEE congress on evolutionary computation conference series [63, 66, 115].

DE implementation is similar to a standard EA. However, in contrast, the DE variants mix the current generation individuals with the scaled differences of randomly selected and distinct individuals. Therefore, there is no need for a separate probability distribution to generate the offspring [32]. The initial iteration of a standard DE algorithm consists of four basic steps: initialization, mutation, recombination or crossover and selection, of which only the last three steps are repeated into the subsequent DE iterations. The iterations continue until a termination criterion (such as exhaustion of maximum function evaluations) is satisfied. Each vector $\vec{x}_{i,G}$ in the current population (called

a target vector at the moment of the reproduction) generates one trial vector $\vec{u}_{i,G}$ using a mutant vector $\vec{v}_{i,G}$. The mutant vector is created applying $\vec{v}_{i,G} = \vec{x}_{r0,G} + F(\vec{x}_{r1,G} - \vec{x}_{r2,G})$, where $\vec{x}_{r0,G}$, $\vec{x}_{r1,G}$, and $\vec{x}_{r2,G}$ are vectors chosen at random from the current population ($r0 \neq r1 \neq r2 \neq i$); $\vec{x}_{r0,G}$ is known as the base vector, $\vec{x}_{r1,G}$, and $\vec{x}_{r2,G}$ are the difference vectors and $F > 0$ is a parameter called scale factor. The trial vector is created by the recombination of the target vector and mutant vector using a crossover probability $CR \in [0, 1]$. In this thesis, a simple version of DE called the DE/rand/1/bin variant is chosen; “rand" indicates how the base vector is chosen, “1" represents how many vector pairs will contribute to differential mutation, and “bin" is the type of crossover (binomial, in our case). In the following sections, we will integrate this baseline DE with change detection, change reaction and constraint handling mechanisms to help us solve DCOPs. A general overview of DE algorithm crafted for dynamic and constrained problems is presented in Algorithm 1.

3.3 Change detection mechanisms

Among the many change detection mechanisms proposed, two approaches are considered in this work: (a) detecting changes by re-evaluating dedicated detectors [86], and (b) detecting changes based on algorithm behaviours.

3.3.1 Re-evaluation of solutions

In the literature, re-evaluation of solutions is the most common change detection approach [86]. The algorithm regularly re-evaluates specific solutions (in this work, it re-evaluates the first and the middle individuals of the population) to detect changes in their function values and/ or the constraints.

3.3.2 Error calculation

Irregularities in algorithm behaviours can also be used to detect changes. In order to do this, we calculate the error after each increase in the evaluations. This error is the difference between the values of the objective function and the optimum values at each time. In minimisation problems, the values of this error should be decreasing over generations. But if a change occurs, this value may not be decreasing anymore. In real world optimisation, incorporating the use of optimal values defeats the purpose of optimising the function in the first place. Contrastingly, the main focus of this work is not to develop and test the performance of the algorithm itself, but to develop and test the

constraint handling techniques that are used. If any differences are detected, then all vectors in the current population are re-evaluated to derive updated values.

3.4 Change reaction mechanisms

As mentioned in Section 2.4, different dynamic handling reaction mechanisms have been proposed in the literature. In this section, we will introduce the diversity handling mechanisms and neural networks (which are used for change reaction mechanism) that we use in the following chapters for our experimental studies.

3.4.1 Diversity promoting techniques

In this section, diversity handling mechanisms are reviewed. Among the many popular niching methods, such as fitness sharing, clearing and species-based, we use standard crowding. The reason for excluding the other niching methods is that they were originally designed for and applied to multi-modal functions. An extensive separate study is needed to apply these methods using a moving peak benchmark (designed for testing multi-modal optimisation in DCOPs) and to investigate the methods thoroughly.

3.4.1.1 Chaos local search

Chaos is a natural phenomenon characterised by randomness and sensitivity to initial conditions. Due to those attributes, chaos has been implemented with success in local searches [57]; such a method was implemented in the case of this mechanism with the purpose of promoting diversity. In our case, chaos only affects the best solution at each iteration to avoid computational complexity and do a fair comparison with other methods. We applied an adaptive dynamic search length that is triggered by change detection.

3.4.1.2 Crowding

From among the many niching methods [1] in the literature, we choose the standard crowding method[108]. According to this method, similar individuals in the population are avoided, creating genotypic diversity [2]. Rather than competition with the parents, the offspring competes with the individual that has the lowest Euclidean distance from it. The crowding distance

[1] Niching techniques are the extension of standard EAs to multi-modal domains

[2] Diversity can be defined at distinct levels; genotypic level refers to differences among individuals over $\vec{x}$ values

operator is a density metric of solutions surrounding a particular solution in the population. It is used to determine the extent of their proximity to other solutions. A solution with a lower crowding distance value implies that the region occupied by this solution is crowded by other solutions. The solutions with a higher crowding distance value are preferred for reproduction. As the problem dimension in Chapter 8 is high and because of the selected crossover rate for DE, in each iteration, the generated offspring is not much different from the parent. This happens because due to a low crossover rate, only a small number of dimensions in the individual will change. In this case, the parent is often the closest individual to the offspring. Thus, we modified the method such that the offspring competes against the N closest individuals.

3.4.1.3 Fitness diversity

While the focus of other methods is on creating genotypic diversity, this method creates phenotypic diversity by avoiding individuals with too close fitness values to each other. The offspring in this method competes with the individual that has the closest fitness value to it [53].

3.4.1.4 No diversity mechanism

This method is a base DE algorithm which uses feasibility rules [35] as its constraint handling technique. There is no explicit method used as its diversity promotion technique. Note that all the other methods use feasibility rules as their constraint handling mechanism.

3.4.1.5 Opposition

This mechanism is based on the estimation of the symmetric opposites of individuals in the population, which leads to find new positions which are closer to the problem optimum [97]. Rahnamayan et al. claim that when solving a problem with several dimensions (and without a priori knowledge), evaluating opposites helps the algorithm in finding fitter individuals. Purely random re-sampling or selection of solutions from a given population increase the chance that the algorithm visit or even revisit unproductive regions of the search space.

3.4.1.6 Random immigrants

This method replaces a certain number of individuals (defined by a parameter 'called replacement rate') with random solutions in the population to assure continuous exploration [46]. In the original paper and the one drew upon it

in Chapter 6, random immigrants are inserted into the population at every generation. In our version (Chapter 8), we consider wall clock timing between each change. Therefore, if we insert solutions at each generation, there is insufficient time for the evolution process and the results are affected adversely. Thus, random solutions are inserted only when a change is detected.

3.4.1.7 Restart population

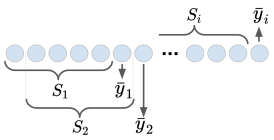
In this method, the population is re-started by random individuals. This is an extreme case of RI that involves considering the replacement rate to the population size.

3.4.1.8 Hyper-mutation

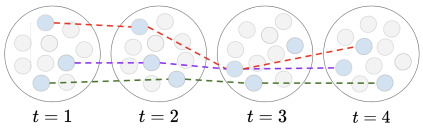
This method was first demonstrated through the use of an adaptive mutation operator in genetic algorithms to solve dynamic constrained optimisation problems [25]. Later, it was used for DE in [5]. After a change detection, some of the DE parameters (CR and F) change for a number of generations, defined empirically (dependent on frequencies of change) to favour larger movements. However, for DE, other mechanisms are needed. This is because when the algorithm is converged, it is not able to promote diversity. Indeed, DE requires population diversity to enhance diversity (see mutant vector Equation in Section 3.2). Therefore, for our version (denoted by HMu), we not only create changes in the DE parameters, but also insert a number of random individuals into the population.

3.4.2 Neural networks

A neural network (NN) is a computing system that learns a function which is a mapping from it's input to outputs. The function is defined by the weight values, connectivity of the network and activation functions of the neurons. In this thesis, the strong approximator, multilayer feedforward artificial NN is adopted. When extracting the change law of a dynamic environment, NN maps the solutions of one environment to the solutions of the next environment [69]. NN consists of multiple layers. Each layer is connected to other layers through multiple neurons and connections. Each neuron has a bias, b, and each connection has a weight, w. The first layer is called an input layer, the last layer is called an output layer, and the other layers are called hidden layers. The output of each layer becomes the input of the following layers.



(A) n_t of previous times are used to predict next optimum



(B) k-best individuals of each time are selected to train NN

FIGURE 3.1: Building samples for neural network

NN is intended to precisely model the optimum movement to make a reliable forecast of the future optimum position. To do this, the best solutions of the previous change periods found by the EA are required to build a time series $(\vec{x_0}, ..., \vec{x_{t-2}}, \vec{x_{t-1}})$ for which the optimum $\vec{x_t}$ of the next change period t has to be predicted (Figure 3.1a). To learn the change pattern of the optimum position, NN will go through a training process. To train the network, k-best individuals (Figure 3.1b: example with $k = 3$) of each time are collected for a number of the previous times (based on a time-window (n_t)). When using NN, it is worth questioning how far back in time (change) an algorithm should search in terms of selecting information on which to form a prediction. In [69], the results of changes in n_t show that the addition of older data introduces noise and misleads the NNs. It is concluded that the accumulation of old data is useful only to extract the overall environmental change information. Therefore, a suggestion is when constructing the training set to select data that has a strong correlation to the predicted targets. In this work, $n_t = 5$ is chosen for the experiments. For future work, the effect of using various time windows (n_t) can be explored. Considering a proper time window, in which the shape of changes has a pattern is effective, as in some real-world problems, the form of the dynamism could change overtime. In addition, another future work is to apply relational NN which prioritise data based on the distance to the predicted value in a time-series prediction (higher priority for closer ones accordingly).

We consider two cases. The first case collects only one best individual ($k = 1$) for five previous time-points ($n_t = 5$) and then predicts the next one (Figure 3.1a). The second procedure considers k-best individuals at each time for $n_t = 5$ and considers a combination of all possibilities (k^{n_t}), in order to build training data (Figure 3.1b). In the latter case, the samples are collected at a faster speed. However, we opt to limit the number of samples collected by choosing a random subset of the above-mentioned combination. This is

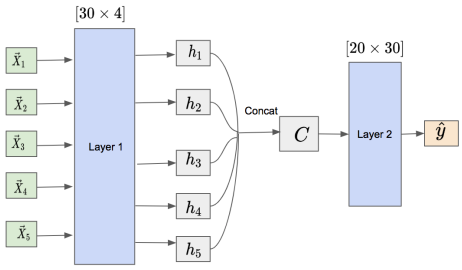


FIGURE 3.2: Structure of neural network

because, if we do not consider limits, the time spent in training data increases exponentially as a result of the large number of samples collected. Also, as we have a sufficiently high number of samples when using $k > 1$, we can limit the NN to use the samples from n_w previous changes. However, where $k = 1$, we keep collecting data, and so do not consider limits for the number of collected samples ($n_w = \infty$). Otherwise, the number of samples would remain too low. It should be noted that, for the first environmental changes, we have a small number of samples. Hence, it is difficult for the NN to generalise from these data. To avoid this situation, we wait until a minimum number of samples are collected in order to train the NN. We call this minimum number of samples 'min_batch size' and empirically assign it a value of 20. When $k = 1$, we have to wait for a large number of time changes before we can begin using the NN. But where $k > 1$, we collect samples at a faster rate, so that the time lag to start using the NN is shorter.

The structure of the applied neural network has two hidden layers (presented in Figure 3.2). The first layer takes as an input an individual position $\vec{x_i}$ with d dimensions and outputs a hidden representation h_i of the individual with four dimensions. As the network uses the best individuals from the past five times to predict the next one (Figure 3.1a), the first layer is applied to each of these five individuals $\vec{x_1}, ..., \vec{x_5}$ independently. As a result, we obtain five hidden representations with four dimensions $h_1, ..., h_5$; to aggregate their information, we choose to concatenate them into a variable H with 4×5 dimensions. The second layer takes H as its input and then outputs a prediction with d dimensions, representing the next best individual. The first layer employs the rectified linear units (ReLU) activation function and the second layer has a linear output without an activation function. To train the network, we use mean squared error as a loss function. The predicted solution or its neighbouring positions can then be used by EA to intensify the search in that region of the solution space. The mechanism to insert the predicted solutions into the population can either entail replacing the worst individuals of the population, or replacing random individuals.

3.5 Constraint handling techniques

EAs lack a mechanism to incorporate the constraints of a given problem into the fitness value of individuals. Thus, many studies have been dedicated to handling the constraints in EAs. Within most successful constraint-handling techniques, the objective function value and the sum of constraint violations (or the constraint violation) are handled separately. The algorithm searches for an optimal solution while balancing the optimisation of the function value and the optimisation of the constraint violation. For a comprehensive survey of constraint handling techniques see [77]. In addition [38] introduces a taxonomy of constraint handling techniques that considers a characterisation of constraints to address black-box and simulation-based optimisation problems. The authors provide formal definitions for several constraint classes and present illustrative examples in the context of the resulting taxonomy. In this section, we only introduce the methods we apply for experimental studies in future chapters.

One important distinction between constraint handling techniques involves the way in which they deal with infeasible solutions. Some techniques (such as penalty function and feasibility rules) are stricter regarding infeasible solutions, while others (such as ϵ-constrained method and stochastic ranking) are more flexible.

3.5.1 Penalty functions

The idea of penalty functions is to transform a constrained optimisation problem into an unconstrained problem. The penalty function can achieve this by adding or subtracting a certain value to or from the objective function based on the extent to which a certain solution violates constraints. Indeed, it tries to decrease the fitness of infeasible solutions in order to favour the selection of feasible solutions. There are different kinds of penalty methods, including static (known as 'death'), dynamic, adaptive, co-evolved and fuzzy-adapted. In this thesis, we only apply a simple penalty method which, for each infeasible solution, considers the following objective function formula [88].

$$\hat{f}(\vec{x}, t) = f(\vec{x}, t) + 2.5\phi(\vec{x}, t) \tag{3.1}$$

The sum of constraint violation $\phi(\vec{x}, t)$ can be calculated as follows:

$$\phi(\vec{x}, t) = \sum_{i=1}^{m} max(0, g_i(\vec{x}, t)) + \sum_{j=1}^{p} |h_j(\vec{x}, t)| \tag{3.2}$$

where $g_i(\vec{x}, t)$ are inequality constraints, i = 1 ... m and $h_j(\vec{x}, t) = 0, j = 1 \dots p$ are equality constraints.

The major drawback of penalty functions is that selecting a proper penalty factor is somewhat difficult without knowing the problem at hand (in our case, on the basis of an empirical experiment, we have chosen a factor of 2.5). If the penalty is too high, for the case where the optimum is at the boundary of the feasible region, the EA will be forced inside the feasible region very quickly. Therefore, it will not be able to move back towards the boundary with the infeasible region. On the other hand, if the penalty is too low, a major part of the search time will be spent to explore the infeasible region, because the penalty will be negligible with respect to the objective function.

3.5.2 Feasibility rules

Feasibility rules are one of the most popular constraint handling techniques used in the context of EAs. This technique (proposed by Deb [35]), consists of a set of three feasibility criteria:

i) Between 2 feasible vectors, the one with the highest fitness value is selected.

ii) If one vector is feasible and the other one is infeasible, the feasible vector is selected.

iii) If both vectors are infeasible, the one with the lowest sum of constraint violation is selected.

The lack of user-defined parameters is one of this method's main advantages. In addition, the rules are simple and flexible, which makes them very suitable for relatively easy combination with any sort of selection mechanism. However, this method may lead to premature convergence [78], since this type of scheme strongly favours feasible solutions. Thus, if no further mechanisms are adopted to preserve diversity (particularly paying attention to the need to keep infeasible solutions in the population), this approach will significantly increase the selection pressure [76].

3.5.3 ϵ-constrained

The ϵ-constrained method was proposed by Takahama et al. in [114]. This method is a type of transformation method that converts an algorithm for unconstrained optimisation into an algorithm for constrained optimisation. This technique has two main elements. The first element is a relaxation of the limit within which a solution is considered feasible, based on its sum of

Algorithm 2 Stochastic Ranking sort algorithm [107].

for $i = 1$ to NP **do**
  for $j = 1$ to $NP - 1$ **do**
    u = random(0,1)
    if $(\phi(\vec{x}_j, t) = \phi(\vec{x_{j+1}}, t)) = 0$ **or** $(u < P_f)$ **then**
      if $f(\vec{x}_j, t) > f(\vec{x_{j+1}}, t)$ **then**
        Swap $\vec{x}_j, t$ with $\vec{x_{j+1}}, t$
      end if
    else
      if $\phi(\vec{x}_j, t) > \phi(\vec{x_{j+1}}, t)$ **then**
        Swap $\vec{x}_j, t$ with $\vec{x_{j+1}}, t$
      end if
    end if
  end for
  if swap not performed **then**
    break
  end if
end for

constraint violation previously defined in equation 5.5, with the aim of using its objective function value as a comparison criterion. The second element is a lexicographical ordering mechanism in which the minimisation of the sum of constraint violation precedes the minimisation of the objective function of a given problem. For any ϵ satisfying $\epsilon \geq 0$, the ϵ level comparisons $<_\epsilon$ and $\leq_\epsilon$ between (f_1, ϕ_1) and (f_2, ϕ_2) are defined in Equation 3.3 and 3.4.

$$(f(\vec{x_1}), \phi(\vec{x_1})) <_\varepsilon (f(\vec{x_2}), \phi(\vec{x_2})) \Leftrightarrow \begin{cases} f(\vec{x_1}) < f(\vec{x_2}), & \text{if } \phi(\vec{x_1}), \phi(\vec{x_2}) \leq \varepsilon \\ f(\vec{x_1}) < f(\vec{x_2}), & \text{if } \phi(\vec{x_1}) = \phi(\vec{x_2}) \\ \phi(\vec{x_1}) < \phi(\vec{x_2}), & \text{otherwise} \end{cases} \tag{3.3}$$

$$(f(\vec{x_1}), \phi(\vec{x_1})) \leq_\varepsilon (f(\vec{x_2}), \phi(\vec{x_2})) \Leftrightarrow \begin{cases} f(\vec{x_1}) \leq f(\vec{x_2}), & \text{if } \phi(\vec{x_1}), \phi(\vec{x_2}) \leq \varepsilon \\ f(\vec{x_1}) \leq f(\vec{x_2}), & \text{if } \phi(\vec{x_1}) = \phi(\vec{x_2}) \\ \phi(\vec{x_1}) < \phi(\vec{x_2}), & \text{otherwise} \end{cases} \tag{3.4}$$

When $\epsilon = 0$,$<_0$ and $\leq_0$ are equivalent to the lexicographic order in which the constraint violation $\phi(\vec{x})$ precedes the function value $f(\vec{x})$. Furthermore, in the case of $\epsilon = \infty$, the ϵ level comparisons $<$ and $\leq$ between function values.

3.5.4 Stochastic ranking

Runarsson and Yao proposed the stochastic ranking (SR) [107]. This technique was designed to deal with the shortcomings of a penalty function (i.e., that neither under-nor over-penalisation represents a good constraint handling technique, and there should be a balance between preserving feasible individuals and rejecting infeasible ones). Rather than being controlled by penalty factors, SR employs a user-defined parameter called P_f, that controls comparison of infeasible solutions: 1) based on their overall constraint violation sum or 2) based only on their objective function value. This technique uses a bubble-sort-like process to rank the solutions in the population, described in the algorithm 2, where I is an individual of the population. $\phi(I_j)$ is the constraint violation sum of the individual I_j. $f(I_j)$ is the objective function value of individual I_j.

3.5.5 Repair methods

Repair methods have shown competitive results compared to other constraint handling methods in constrained optimisation. Particularly, repair methods have some attributes which would make them a successful candidate to solve DCOPs. First, the operation of the repair methods do not conflict with the operation of dynamic handling strategies. Some constraint handling techniques may not work well with some dynamic handling mechanisms, such as diversity-maintaining or introducing strategies. This is because, these strategies select individuals based on their feasibility, and feasible individuals might have a different probability of selection than infeasible individuals. Such a bias in selection might cause many diversified individuals to be discarded because of their infeasibility. Repair methods somewhat prohibit this drawback, because they accept both feasible and infeasible individuals in the same way. In other words, it does not care about the feasibility of an individual, given that this individual can provide a high quality repaired solution.

Second, repair methods are naturally ideal for tracking the moving feasible region. In the repair operation, the repaired individuals will always be closer to existing reference individuals than the original individual. As a result of that operation, if the algorithm is able to have at least one reference individual in the moving feasible region, the repair method will have a chance of sending more individuals toward that reference individual. Consequently, the repair method will be capable of tracking that moving region. Third, the repair method intrinsically supports elitism because the best found feasible solutions

Algorithm 3 Dynamic differential evolution (DDE) with repair methods

```
Create and evaluate a randomly initial population x⃗_{i,G}, i = 1, ..., NP
for G ← 1 to MAX_GEN do
  for i ← 1 to NP do
    Change detection mechanism (x⃗_{i,G})
    Randomly select r0 ≠ r1 ≠ r2 ≠ i
    J_rand = randint[1, D]
    for j ← 1 to D do
      if rand_j ≤ CR Or j = J_rand then
        u_{i,j,G} = x_{r1,j,G} + F(x_{r2,j,G} − x_{r3,j,G})
      else
        u_{i,j,G} = x_{i,j,G}
      end if
    end for
    if u_{i,j,G} is infeasible then
      Use the repair method
    end if
    if f(u⃗_{i,G}) ≤ f(x⃗_{i,G}) then
      x⃗_{i,G+1} = u⃗_{i,G}
    else
      x⃗_{i,G+1} = x⃗_{i,G}
    end if
  end for
end for
```

will always be stored in the reference population. This feature enables the method to maintain diversity effectively in DCOPs.

The main idea of a repair method is to use a transformation process to convert an infeasible solution into a feasible one. However, unlike other constraint handling techniques, this method does not require special operators or any modifications of the fitness function. In some repair methods, reference feasible solutions are required [79, 87, 94, 95]. However, the repair methods presented in [4] and [18] does not require feasible reference solutions. Repair methods used in DCOPs have had an important role in the algorithm's recovery after a change, since they help to move the infeasible solutions toward the feasible region. In the related literature on DCOPs, there have been four repair methods utilised for constraint handling, which we present below. A general overview of a DE algorithm using repair methods is presented in Algorithm 3.

3.5.5.1 Reference-based repair method

This method was originally proposed in [79], and [87] who utilises this method with a simple genetic algorithm to solve DCOPs. In this method, a reference feasible population (R) is first created. If an individual of the search population (S) is infeasible, a new individual is generated on the straight line joining the

Algorithm 4 Reference-based and offspring-repair methods

Require: $\vec{u}_{i,G}$ {trial vector}
  $counter = 0$
2: **while** $\vec{u}_{i,G}$ is infeasible and $counter \leq$ RL **do**
    Select the reference individual $r \in R$ based on:
4: $\begin{cases} \text{Randomly} & <\text{reference-based}> \\ \text{Min distance between } \vec{u}_{i,G} \text{ and } r & <\text{offspring}> \end{cases}$
6: Create random number $a = U[0,1]$
    Create a new individual in the segment between $\vec{u}_{i,G}$ ($s \in S$) and r
8: $\vec{u}_{i,G} = a \cdot r + (1-a) \cdot \vec{u}_{i,G}$
    if $\vec{u}_{i,G}$ is infeasible **then**
10: go to step 2
    else
12: Update reference population if the repaired solution has better fitness value than R
    end if
14: $counter = counter + 1$
  end while
16: Return $\vec{u}_{i,G}$

infeasible solution and a randomly chosen member of R. This process will continue until the infeasible solution is repaired or a repair limit (RL=100) is computed. If the new feasible solution has a better fitness value, it will be replaced by the selected reference individual. An overview of this method used for our investigations is presented in Algorithm 4.

3.5.5.2 Offspring-repair method

This method was applied to DCOPs in [94, 95]. Within this method, a reference feasible population (R) is generated. For any infeasible solution of the search population (S), a new individual is generated on the straight line joining the infeasible solution and the nearest member of the reference population R, based on Euclidean distance. This process will continue until the infeasible solution is repaired or a repair limit (RL=100) is computed. If the new feasible solution has a better fitness value, it will be replaced by the selected reference individual. This method is similar to the reference-based repair method [87], with the only difference being the process of selecting the reference solution. An overview of this method is presented in Algorithm 4.

3.5.5.3 Mutant-repair method

The mutant-repair method (see Algorithm 5) is based on the differential mutation operator, and does not require reference solutions [4]. For each infeasible solution, three new and temporal solutions are generated at random,

Algorithm 5 Mutant-repair method

Require: $\vec{u}_{i,G}$ {trial vector}
  $counter = 0$
while $\vec{u}_{i,G}$ is infeasible and $counter \leq$ RL **do**
  Generate three random vectors ($\vec{u}_{r0,G}$, $\vec{u}_{r1,G}$ and $\vec{u}_{r2,G}$)
  $\vec{u}_{i,G} = \vec{u}_{r0,G} + F(\vec{u}_{r1,G} - \vec{u}_{r2,G})$
  $counter = counter + 1$
end while
Return $\vec{u}_{i,G}$

Algorithm 6 Gradient-based repair method

Require: $\vec{u}_{i,G}$ {trial vector}
  $counter = 0$
while $\vec{u}_{i,G}$ is infeasible and $counter \leq$ RL **do**
  Calculate the constraint violation
  Calculate the amount of solution movement $\Delta\vec{u}_{i,G}$ based on the current constraint violation and the gradient information
  $\vec{u}_{i,G} = \vec{u}_{i,G} + \Delta\vec{u}_{i,G}$
  $counter = counter + 1$
end while
Return $\vec{u}_{i,G}$

and a differential mutation operator similar to the one used in DE is applied. This repair method is applied until the infeasible solution is repaired or a specific number of unsuccessful trials to obtain a feasible solution have been carried out (RL).

3.5.5.4 Gradient-based repair method

The gradient-based repair method (see Algorithm 6) was first applied in a simple GA [23] to handle constraints in a static optimisation problem, and in [18], it was applied to the solving of DCOPs. In this method, gradient information pertaining to the constraints is utilised to repair the infeasible solutions [23]. For this purpose, the gradient of the constraints based on the solution vector (which represent the rate of change of constraints based on each variable) is calculated. At the next step, the constraint violations are calculated. Based on this calculation and the vector of the gradient, the solutions move toward the feasible region with the proportional quantity. The constraints that are non-violated are not considered in these calculations. The main idea of this method is to only change the effective variables over constraints that have a violation. More detail about this method can be found in [18].

Chapter 4

Constraint Handling Techniques in Dynamic Constrained Continuous Problems

4.1 Introduction

There are studies about the effect of constrained handling techniques in static optimisation problems. However, when dealing with DCOPs, there is not a substantial study in the behaviour of common constraint handling techniques. In this section, we conduct two distinct experiments. First, we study four most commonly used constraint handling techniques (stochastic ranking, ϵ-constrained, penalty and feasibility rules) with DE to observe the behaviour of these techniques. Second, we conduct an experiment pertaining repair methods separately. The reason to conduct a separate study for repair methods from other constraint handling techniques is that repair methods often require a noticeable number of extra fitness evaluations compared to standard methods. This is because the repair methods need to create reference population, as well as evaluate the fitness for each repaired solution.

For the purpose of analysis, we use a common benchmark to determine which techniques are suitable for the most prevalent types of DCOPs. In addition, common measures in static environments are adapted to suit dynamic environments. While an overall superior technique could not be determined, certain techniques outperformed others in different aspects such as rate of optimisation or reliability of solutions.

Existing algorithms already find it difficult to optimize static constrained problems, and it becomes even more difficult when constraints are dynamically changing [87]. There currently exists a substantial amount of research

into dynamic unconstrained optimisation [86] and static constrained optimisation [77] for EAs. However, this is not the case for dynamic constrained optimisation. One of the most important aspects of solving DCOPs is using an effective constraint handling technique to deal with the dynamic constraints in order to guide the search to those regions with feasible solutions and quickly adapt if constraints are changing. In the specialized literature about DCOPs, the constraint handling techniques that have been applied include penalty function [85], repair methods [19, 87, 94] and feasibility rules [5].

While these methods show sound results for applying in DCOPs, other methods like ϵ-constrained [114] and stochastic ranking [107] due to their characteristics seem to have competitive results in DCOPs. These characteristics mostly relate to the ability of the constraint handling method to increase or maintain diversity in the balance of feasible and infeasible solutions of the population. A comprehensive survey about the details of constraint handling techniques used with EAs can be found in [77]. In ϵ-constrained, the infeasible solutions are treated more mildly compared to feasibility rules, which implies that a higher diversity is usually maintained. Similarly, stochastic ranking ranks the solutions not only based on the objective values and the feasibility of the solutions, but also a stochastic behaviour is seen in the algorithm selection. This implies that infeasible solutions close to the region of feasibility are maintained in the population, which may help when constraints change. In the remainder of this chapter, firstly, we present the results of the experiment with common constraint handling techniques. Afterwards, we introduce the experiment results for repair methods.

4.2 Standard constraint handling techniques

In this section, we investigate common constrained handling techniques, introduced in Chapter 3, including stochastic ranking, ϵ-constrained, penalty and feasibility rules. As mentioned earlier, we consider repair methods in a separate study (Section 4.3) because they are mechanisms that apply special operators to transform solutions [77]. They use extra evaluations during the optimisation procedure compared to the standard constraint handling techniques [3], this provides an unfair advantage in the results due to repair methods' ability to optimise faster and increase performance dramatically.

Briefly, the results show based on the offline error, feasibility and epsilon outperform the other techniques and maintain competitive performance with each other. However, the other techniques are more suited for alternative measures. Stochastic severely outperforms all other techniques in terms of

Algorithm 7 Differential Evolution Algorithm (DE/rand/1/bin)

```
G=0
Create a randomly initial population x⃗_{i,G}, for i = 1,...,NP
Evaluate f(x⃗_{i,G}), for i = 1,...,NP
for G ← 1 to MAX_GEN do
    for i ← 1 to NP do
        if i = 1 or i = NP/2 then
            Change detection mechanism (x⃗_{i,G})
        end if
        Randomly select r0 ≠ r1 ≠ r2 ≠ i
        J_rand = randint[1, D]
        for j ← 1 to D do
            if rand_j ≤ CR Or j = J_rand then
                u_{i,j,G} = x_{r1,j,G} + F(x_{r2,j,G} − x_{r3,j,G})
            else
                u_{i,j,G} = x_{i,j,G}
            end if
        end for
        Select u_{i,j,G} or x_{i,j,G} based on the constraint handling technique
    end for
    G = G + 1
end for
```

speed, it makes up for its lack of reliability in how few evaluations it requires to find an optimum solution. While penalty is not the fastest nor does it have the least number of constraint violations, it is the most reliable of all the techniques and frequently returns the greatest number of successful solutions, considering the proposed measure (convergence score). Stochastic is also the highest performing technique for static constraints. However in the dynamic constraints, the techniques struggle to find successful solutions in the given time frame. A suggested solution to this issue is the addition of mechanisms to increase diversity or repair solutions to increase feasibility.

4.2.1 Experimental design

For ϵ-constrained method, the value of T_c is used in order to change the value of ϵ after a known number of iterations. The chosen benchmark originally has 18 test problems [87] (see Chapter 2 for details of the benchmark). However, in this section, only constrained problems were used for the experiments, consisting of 14 test problems. In this benchmark, the test problems consist of a variety of characteristics such as i) disconnected feasible regions (1-3), ii) the global optima at the constraints' boundary or switchable between disconnected regions, or iii) the different shape and percentage of feasible area. In the experiments, for the objective function, only medium severity is considered ($k = 0.5$), while different change severities are considered for the

TABLE 4.1: Average and standard deviation of modified offline error values. Best results are remarked in boldface.

Algorithms	$S = 10$						
	G24_3	**G24_3b**	**G24_4**	**G24_5**	**G24_7**		
Epsilon	0.177(±0.022)	0.23(±0.026)	0.232(±0.028)	0.223(±0.031)	0.362(±0.054)		
Feasibility	**0.165(±0.022)**	**0.227(±0.024)**	**0.23(±0.03)**	**0.216(±0.083)**	**0.298(±0.06)**		
Penalty	0.235(±0.06)	0.491(±0.225)	0.628(±0.297)	2.316(±1.521)	1.51(±0.479)		
Stochastic	0.219(±0.047)	0.254(±0.078)	0.231(±0.057)	0.392(±0.118)	0.457(±0.124)		
Algorithms	$S = 20$						
	G24_1	**G24_f**	**G24_2**	**G24_3**	**G24_3b**	**G24_3f**	**G24_4**
Epsilon	**0.25(±0.04)**	**0.028(±0.011)**	0.101(±0.014)	0.179(±0.037)	0.289(±0.026)	**0.028(±0.011)**	0.282(±0.039)
Feasibility	0.266(±0.05)	0.032(±0.019)	**0.097(±0.017)**	**0.148(±0.015)**	0.276(±0.03)	0.077(±0.258)	**0.273(±0.033)**
Penalty	0.714(±0.392)	0.035(±0.024)	1.142(±0.977)	0.197(±0.06)	0.706(±0.297)	0.063(±0.039)	0.729(±0.357)
Stochastic	0.289(±0.062)	0.227(±0.143)	0.123(±0.04)	0.203(±0.044)	**0.258(±0.046)**	0.132(±0.15)	0.277(±0.056)
	G24_5	**G24_6a**	**G24_6b**	**G24_6c**	**G24_6d**	**G24_7**	**G24_8b**
Epsilon	0.158(±0.022)	0.122(±0.037)	0.087(±0.012)	0.1(±0.03)	0.143(±0.04)	0.264(±0.034)	0.285(±0.039)
Feasibility	**0.141(±0.017)**	0.105(±0.024)	**0.082(±0.012)**	**0.089(±0.021)**	0.169(±0.062)	0.247(±0.029)	**0.276(±0.034)**
Penalty	1.955(±1.349)	0.247(±0.079)	0.213(±0.075)	0.284(±0.077)	0.141(±0.053)	0.704(±0.153)	0.612(±0.095)
Stochastic	0.162(±0.04)	**0.091(±0.022)**	0.111(±0.029)	0.103(±0.027)	**0.138(±0.039)**	**0.204(±0.053)**	0.457(±0.118)
Algorithms	$S = 50$						
	G24_3	**G24_3b**	**G24_4**	**G24_5**	**G24_7**		
Epsilon	0.174(±0.036)	0.286(±0.031)	0.282(±0.035)	**0.13(±0.021)**	0.193(±0.029)		
Feasibility	**0.1(±0.018)**	0.257(±0.05)	0.241(±0.03)	0.135(±0.023)	**0.188(±0.03)**		
Penalty	0.122(±0.04)	0.698(±0.374)	0.718(±0.382)	1.494(±1.243)	0.385(±0.09)		
Stochastic	0.127(±0.032)	**0.226(±0.043)**	**0.238(±0.042)**	0.146(±0.045)	0.24(±0.127)		

constraints ($S = 10$, 20 and 50). Based on the definition of the constrains in this benchmark [87], $S = 10$, $S = 20$ and $S = 50$ represent the severity of the changes on the constraints. The frequency of change (f_c) is considered equal to 1000 evaluations (only in the objective function). The configurations for the experiments are as follows. The number of runs in the experiments are 30, and the number of considered times for dynamic perspective of the test algorithm is $5/k$ ($k = 0.5$). Parameters relating to DE algorithm are as follows: DE variant is DE/rand/1/bin, population size is 20, $F \sim \mathcal{U}(0.2, 0.8)$, crossover probability is 0.2, and the maximum number of evaluations is a multiplication of frequency and time: $f_c \cdot 5/k$. These parameters have been chosen in a set of primarily experiments. We discard to bring the relevant experiments to maintain the focus of this section on comparing the different constraint handling techniques rather than the behaviour of the DE algorithm. In the experiments, four approaches including ϵ-constrained, feasibility rules, penalty function and stochastic ranking, as explained in Chapter 3, have been applied for handling the constraints in the DE algorithm.

4.2.2 Experimental analysis

In the analysis, the effects of different severities on the constraints are considered for these fourteen test problems. We do not bring the results for changes of frequency since frequency does not have any effect in the behaviour of the constraint handling techniques.

TABLE 4.2: Statistical tests on the offline error values in Table 2. "$X^{(-)}$" means that the corresponding algorithm outperformed algorithm X. "$X^{(+)}$" means that the corresponding algorithm was dominated by algorithm X. If algorithm X does not appear in column Y means no significant differences between X and Y.

Functions	$S = 10$			
	Epsilon(1)	**Feasibility(2)**	**Penalty(3)**	**Stochastic(4)**
G24_3 (7.1-49.21%)	$3^{(-)}, 4^{(-)}$	$3^{(-)}, 4^{(-)}$	$1^{(+)}, 2^{(+)}$	$1^{(+)}, 2^{(+)}$
G24_3b (7.1-49.21%)	$3^{(-)}$	$3^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$3^{(-)}$
G24_4 (0-44.2%)	$3^{(-)}$	$3^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$3^{(-)}$
G24_5 (0-44.2%)	$3^{(-)}, 4^{(-)}$	$3^{(-)}, 4^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$1^{(+)}, 2^{(+)}, 3^{(-)}$
G24_7 (0-44.2%)	$3^{(-)}$	$3^{(-)}, 4^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$2^{(+)}, 3^{(-)}$
Functions	$S = 20$			
	Epsilon(1)	**Feasibility(2)**	**Penalty(3)**	**Stochastic(4)**
G24_1 (44.2%)	$3^{(-)}$	$3^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$3^{(-)}$
G24_f (44.2%)	$4^{(-)}$	$4^{(-)}$	$4^{(-)}$	$1^{(+)}, 2^{(+)}, 3^{(+)}$
G24_2 (44.2%)	$3^{(-)}$	$3^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$3^{(-)}$
G24_3 (7.1-49.21%)	$2^{(+)}$	$1^{(-)}, 3^{(-)}, 4^{(-)}$	$2^{(+)}$	$2^{(+)}$
G24_3b (7.1-49.21%)	$3^{(-)}$	$3^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$3^{(-)}$
G24_3f (7.1%)	$3^{(-)}, 4^{(-)}$	$3^{(-)}, 4^{(-)}$	$1^{(+)}, 2^{(+)}$	$1^{(+)}, 2^{(+)}$
G24_4 (0-44.2%)	$3^{(-)}$	$3^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$3^{(-)}$
G24_5 (0-44.2%)	$3^{(-)}$	$3^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$3^{(-)}$
G24_6a (16.68%)	$3^{(-)}, 4^{(+)}$	$3^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$1^{(-)}, 3^{(-)}$
G24_6b (50.01%)	$3^{(-)}, 4^{(-)}$	$3^{(-)}, 4^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$1^{(+)}, 2^{(+)}, 3^{(-)}$
G24_6c (33.33%)	$3^{(-)}$	$3^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$3^{(-)}$
G24_6d (20.91%)	-	$3^{(+)}, 4^{(+)}$	$2^{(-)}$	$2^{(-)}$
G24_7 (0-44.2%)	$3^{(-)}, 4^{(+)}$	$3^{(-)}, 4^{(+)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$1^{(-)}, 2^{(-)}, 3^{(-)}$
G24_8b (44.2%)	$3^{(-)}, 4^{(-)}$	$3^{(-)}, 4^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$1^{(+)}, 2^{(+)}, 3^{(-)}$
Functions	$S = 50$			
	Epsilon(1)	**Feasibility(2)**	**Penalty(3)**	**Stochastic(4)**
G24_3 (7.1-49.21%)	$2^{(+)}, 3^{(+)}, 4^{(+)}$	$1^{(-)}, 4^{(-)}$	$1^{(-)}$	$1^{(-)}, 2^{(+)}$
G24_3b (7.1-49.21%)	$3^{(-)}, 4^{(+)}$	$3^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$1^{(-)}, 3^{(-)}$
G24_4 (0-44.2%)	$2^{(+)}, 3^{(-)}, 4^{(+)}$	$1^{(-)}, 3^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$1^{(-)}, 3^{(-)}$
G24_5 (0-44.2%)	$3^{(-)}$	$3^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$3^{(-)}$
G24_7 (0-44.2%)	$3^{(-)}$	$3^{(-)}$	$1^{(+)}, 2^{(+)}, 4^{(+)}$	$3^{(-)}$

4.2.2.1 Analysis I: performance measure

The results obtained for the four constraint handling techniques using modified offline error are summarized in Table 4.1. Furthermore, for the statistical validation, the 95%-confidence Kruskal-Wallis test and the Bonferroni post hoc test, as suggested in [37] are presented (see Table 4.2). Non-parametric tests were adopted because the samples of runs did not fit to a normal distribution based on the Kolmogorov-Smirnov test. Worth to mention that we removed the functions G24_1, G24_f, G24_2, G24_3f, G24_6a, G24_6b, G24_6c, G24_6d and G24_8b from severity S=10 and 50 because they have static constraints. Therefore we include the results for these functions only for severity S=20 since they are the same for other severities as well.

Table 4.1, illustrates the modified offline error values for different functions separated for each severity. From this table, one immediate conclusion is that penalty performed the worst among all techniques based on modified

offline error values as it has higher error values for almost all of the functions, regardless of severity. However, to observe whether the methods have significant differences or not, the Kruskal-Wallis test has been carried out and the results are presented in Table 4.2. The results of the statistical tests can be summarized as following observations: in static constraint function G24_6d penalty performed better than feasibility, and in function G24_f it outperformed stochastic for severity S=20. In dynamic constraint function G24_3, for severity S=50 it outperformed epsilon.

Among the techniques, epsilon and feasibility showed similar results. This is because epsilon uses a modification of feasibility rules, thus they have a similar trend to handling the constraints. This causes the two to lack significant difference in almost all of the functions excluding G24_3 (for S=20 and 50) and G24_4 (for S=50) where epsilon is the better performing technique.

In regards to stochastic, it was outperformed by both epsilon and feasibility in some functions like G24_3, G24_5 (S=10), G24_f, G24_3f, G24_6b, G24_8b (S=20) however, it only had significant difference with feasibility and not epsilon in functions G24_7 (S=10) and G24_3 (S=20 and 50).

In general, severity did not have any significant effect on the results. For testing other characteristics of the constraint handling techniques like feasibility probability, convergence rate, average number of function evaluations required for finding the first successful solution, convergence score and progress ratio to determine which performs the most effectively, other measures are defined and analysed in the next section.

4.2.2.2 Analysis II: behaviour measures

Tables 4.3, and 4.4 show the result of the measurements that were defined in Section 4.2.1. General observations regarding to the algorithms' behaviour in these measures are summarized as follows.

Due to the lower rate of success (SR_t) in the stochastic ranking, this technique also tends to not find the optimum solution more often than its counterparts as shown in G24_f and G24_3f. This is attributed by the random nature of the stochastic ranking and its lack of consistent reliability as shown in all functions with non-zero success rates (SR_t).

Due to the large area of feasibility in this benchmark, the constraint handling techniques tended to have very high if not perfect feasibility rates (FP_t), however the penalty technique showed lower feasibility rates than its counterparts due to its nature of accepting infeasible solutions during optimisation.

Based on the three measurements (CS_t, AE_t and SR_t) in dynamic constraint functions including G24_3, G24_3b, G24_4, G24_7, with the exception of G24_5, when the severity of the constraints is equal to 20 and 50, it is harder for the constraint handling techniques to converge with the optimal solutions. Conversely, for s=10, the constraint handling techniques are unable to converge to optimal solution for function G24_7. Although this trend is also true for the static constraint function G24_1.

For all of the functions, the three constraint handling techniques (epsilon, feasibility and penalty) had near identical success rates (SR_t), while not exactly the same they fell within one standard deviation of each other. However, the stochastic ranking technique had vastly different success rates compared to its counterparts.

Larger values for the progress ratio (PR_t) do not always indicate better performance, since it depends on the distance between the first feasible solution and the best solution found. Even if the distance between these solutions is large, the best solution found can be stuck in a local optimum and could never reach the global optimum. Indeed, the calculation of this measure does not take optimum values into consideration.

Improvement of the constraint handling techniques would require additional optimisation mechanisms as these techniques have very small standard deviation in the rate of optimisation leading to similar progress ratio values.

By gauging the performance of the constraint handling methods using this extensive list of measures, it has allowed an in-depth analysis of the nuances and specific behaviours of each algorithm and how they compare to each other. Simply analyzing the difference in fitness between them only works until they reach the same solution.

4.2.3 Conclusion and discussions

In this section, we have compared common constraint handling techniques for solving DCOPs. For the measurements, a modified version of offline error and other measures including average evaluations, convergence score, progress ratio, feasibility ratio and successful ratio were adapted for dynamic environments and used for different constraint's change severity. While the modified offline error data revealed competitive results between epsilon and feasibility, stochastic was considerably less reliable with large variations in the results and penalty presents the worst performance in terms of this measurement. However, stochastic managed the constraints and guided the algorithm to a successful solution much faster than any other technique, albeit,

with a considerably lower reliability. This would make stochastic the more effective choice for simpler optimisation problems, where reliability is not an important factor in the performance. Conversely, penalty is the most reliable of the techniques which makes up for its lack of speed in constraint management. It takes far longer than the other techniques to reach a feasible solution, but, it consistently finds more successful solutions overall. Taking the proposed measure (convergence score) into consideration, stochastic compensates for its unreliability with its speed and frequently scores the best out of the techniques in functions with static constraints. While this may be the case, in the functions with dynamic constraints, all of the techniques struggled to find successful solutions in the given time frame. This problem can be mitigated by adding additional mechanisms to the algorithms that increase its performance like methods of increasing diversity of solutions or repairing infeasible solutions. In the future of dynamic constrained optimisation, new constraint handling techniques would need to be developed to deal with the dynamic nature of the problem.

TABLE 4.3: Average and standard deviation of average evaluations (AE_t), convergence score (CS_t), progress ratio (PR_t), feasibility ratio(FR_t), successful ratio (SR_t). For s=10 and 50 only the functions that have dynamic constraints are displayed. Best results are remarked in boldface.

s = 10

Measures	G24_3				G24_3b				G24_4			
	Epsilon	Feasibility	Penalty	Stochastic	Epsilon	Feasibility	Penalty	Stochastic	Epsilon	Feasibility	Penalty	Stochastic
AE_t	766.13$^{\pm(102.98)}$	709.17$^{\pm(164.94)}$	768.50$^{\pm(239.98)}$	**329.33**$^{\pm(244.60)}$	354.17$^{\pm(126.91)}$	340.17$^{\pm(128.77)}$	340.67$^{\pm(93.10)}$	**145.23**$^{\pm(114.67)}$	382.10$^{\pm(128.28)}$	302.43$^{\pm(142.03)}$	330.70$^{\pm(102.17)}$	**193.93**$^{\pm(103.79)}$
CS_t	**3895.59**$^{\pm(440.57)}$	4255.00$^{\pm(2006.64)}$	4116.96$^{\pm(3509.32)}$	4490.91$^{\pm(901.67)}$	1264.88$^{\pm(400.47)}$	1308.33$^{\pm(493.97)}$	1148.31$^{\pm(400.45)}$	**871.40**$^{\pm(522.78)}$	1432.88$^{\pm(434.23)}$	1226.08$^{\pm(480.36)}$	1127.39$^{\pm(533.76)}$	**1077.41**$^{\pm(531.86)}$
PR_t	**1.09**$^{\pm(0.00)}$	**1.09**$^{\pm(0.00)}$	**1.09**$^{\pm(0.00)}$	1.07$^{\pm(0.02)}$	**1.25**$^{\pm(0.08)}$	1.21$^{\pm(0.05)}$	1.12$^{\pm(0.09)}$	1.21$^{\pm(0.05)}$	**1.23**$^{\pm(0.06)}$	1.21$^{\pm(0.05)}$	1.15$^{\pm(0.07)}$	1.21$^{\pm(0.05)}$
FP_t	**0.83**$^{\pm(0.00)}$	**0.83**$^{\pm(0.00)}$	**0.83**$^{\pm(0.00)}$	0.82$^{\pm(0.01)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.77$^{\pm(0.07)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.75$^{\pm(0.75)}$	**1.00**$^{\pm(0.00)}$
SR_t	**0.20**$^{\pm(0.02)}$	0.17$^{\pm(0.05)}$	0.19$^{\pm(0.03)}$	0.07$^{\pm(0.06)}$	0.28$^{\pm(0.04)}$	0.26$^{\pm(0.06)}$	**0.30**$^{\pm(0.02)}$	0.17$^{\pm(0.08)}$	0.27$^{\pm(0.05)}$	0.25$^{\pm(0.05)}$	**0.29**$^{\pm(0.02)}$	0.18$^{\pm(0.06)}$
Measures	G24_5				G24_7							
	Epsilon	Feasibility	Penalty	Stochastic	Epsilon	Feasibility	Penalty	Stochastic				
AE_t	128.70$^{\pm(67.83)}$	124.63$^{\pm(67.32)}$	133.00$^{\pm(41.09)}$	**49.50**$^{\pm(56.80)}$	NaN	NaN	NaN	NaN				
CS_t	270.00$^{\pm(132.54)}$	261.47$^{\pm(132.13)}$	271.43$^{\pm(84.44)}$	**126.92**$^{\pm(226.41)}$	NaN	NaN	NaN	NaN				
PR_t	**0.43**$^{\pm(0.05)}$	0.42$^{\pm(0.04)}$	0.26$^{\pm(0.03)}$	0.40$^{\pm(0.07)}$	**0.59**$^{\pm(0.02)}$	0.59$^{\pm(0.05)}$	0.50$^{\pm(0.03)}$	0.55$^{\pm(0.03)}$				
FP_t	0.91$^{\pm(0.03)}$	**0.91**$^{\pm(0.02)}$	0.71$^{\pm(0.71)}$	0.89$^{\pm(0.06)}$	0.92$^{\pm(0.04)}$	**0.94**$^{\pm(0.05)}$	0.59$^{\pm(0.59)}$	0.89$^{\pm(0.09)}$				
SR_t	0.48$^{\pm(0.04)}$	0.48$^{\pm(0.04)}$	**0.49**$^{\pm(0.03)}$	0.39$^{\pm(0.12)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$				

s = 50

Measures	G24_3				G24_3b				G24_4			
	Epsilon	Feasibility	Penalty	Stochastic	Epsilon	Feasibility	Penalty	Stochastic	Epsilon	Feasibility	Penalty	Stochastic
AE_t	NaN	NaN	NaN	NaN	NaN	NaN	**31.47**$^{\pm(0.00)}$	NaN	NaN	NaN	NaN	NaN
CS_t	NaN	NaN	NaN	NaN	NaN	NaN	**9440.00**$^{\pm(0.00)}$	NaN	NaN	NaN	NaN	NaN
PR_t	0.88$^{\pm(0.00)}$	0.88$^{\pm(0.00)}$	**0.89**$^{\pm(0.01)}$	0.87$^{\pm(0.01)}$	**0.98**$^{\pm(0.06)}$	0.97$^{\pm(0.05)}$	0.92$^{\pm(0.14)}$	0.98$^{\pm(0.09)}$	0.96$^{\pm(0.07)}$	0.96$^{\pm(0.05)}$	0.88$^{\pm(0.13)}$	**1.00**$^{\pm(0.09)}$
FP_t	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.67$^{\pm(0.12)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.75$^{\pm(0.12)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.70$^{\pm(0.17)}$	**1.00**$^{\pm(0.00)}$
SR_t	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	**0.00**$^{\pm(0.02)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$
Measures	G24_5				G24_7							
	Epsilon	Feasibility	Penalty	Stochastic	Epsilon	Feasibility	Penalty	Stochastic				
AE_t	119.10$^{\pm(79.74)}$	126.10$^{\pm(70.38)}$	149.73$^{\pm(39.45)}$	**34.13**$^{\pm(38.16)}$	NaN	NaN	NaN	NaN				
CS_t	255.21$^{\pm(157.67)}$	266.41$^{\pm(138.21)}$	303.51$^{\pm(89.38)}$	**86.05**$^{\pm(116.57)}$	NaN	NaN	NaN	NaN				
PR_t	0.70$^{\pm(0.04)}$	**0.71**$^{\pm(0.03)}$	0.70$^{\pm(0.09)}$	0.71$^{\pm(0.06)}$	**0.97**$^{\pm(0.00)}$	0.97$^{\pm(0.01)}$	0.96$^{\pm(0.02)}$	0.95$^{\pm(0.02)}$				
FP_t	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.87$^{\pm(0.09)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.74$^{\pm(0.14)}$	**1.00**$^{\pm(0.00)}$				
SR_t	0.47$^{\pm(0.05)}$	0.47$^{\pm(0.04)}$	**0.49**$^{\pm(0.02)}$	0.40$^{\pm(0.08)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$				

TABLE 4.4: Average and standard deviation of average evaluations (AE_t), convergence score (CS_t), progress ratio (PR_t), feasibility ratio(FR_t), successful ratio (SR_t). For s=20 all the 14 functions are displayed. Best results are remarked in boldface.

s = 20

Measures	G24_1				G24_f				G24_2			
	Epsilon	Feasibility	Penalty	Stochastic	Epsilon	Feasibility	Penalty	Stochastic	Epsilon	Feasibility	Penalty	Stochastic
AE_t	NaN	NaN	NaN	NaN	72.30$^{\pm(35.80)}$	**66.17**$^{\pm(39.24)}$	126.30$^{\pm(76.77)}$	NaN	47.50$^{\pm(30.33)}$	41.90$^{\pm(20.01)}$	41.07$^{\pm(26.55)}$	**33.70**$^{\pm(20.21)}$
CS_t	NaN	NaN	NaN	NaN	98.14$^{\pm(45.15)}$	**89.41**$^{\pm(51.01)}$	184.83$^{\pm(155.89)}$	NaN	79.17$^{\pm(50.55)}$	69.83$^{\pm(33.34)}$	68.44$^{\pm(44.25)}$	**68.31**$^{\pm(37.12)}$
PR_t	**1.12**$^{\pm(0.01)}$	**1.12**$^{\pm(0.01)}$	1.01$^{\pm(0.16)}$	1.09$^{\pm(0.02)}$	**1.03**$^{\pm(0.00)}$	**1.03**$^{\pm(0.00)}$	**1.03**$^{\pm(0.00)}$	1.01$^{\pm(0.02)}$	**0.59**$^{\pm(0.00)}$	**0.59**$^{\pm(0.00)}$	0.57$^{\pm(0.04)}$	0.57$^{\pm(0.01)}$
FP_t	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.72$^{\pm(0.15)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.79$^{\pm(0.13)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.89$^{\pm(0.11)}$	**1.00**$^{\pm(0.00)}$
SR_t	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	**0.74**$^{\pm(0.05)}$	0.74$^{\pm(0.06)}$	0.68$^{\pm(0.12)}$	0.00$^{\pm(0.00)}$	**0.60**$^{\pm(0.00)}$	**0.60**$^{\pm(0.00)}$	**0.60**$^{\pm(0.00)}$	0.49$^{\pm(0.09)}$
Measures	G24_3				G24_3b				G24_3f			
	Epsilon	Feasibility	Penalty	Stochastic	Epsilon	Feasibility	Penalty	Stochastic	Epsilon	Feasibility	Penalty	Stochastic
AE_t	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	72.73$^{\pm(36.65)}$	**58.50**$^{\pm(40.80)}$	93.47$^{\pm(56.75)}$	NaN
CS_t	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	98.73$^{\pm(46.59)}$	**83.57**$^{\pm(51.31)}$	136.78$^{\pm(95.69)}$	NaN
PR_t	0.96$^{\pm(0.01)}$	0.96$^{\pm(0.00)}$	**0.97**$^{\pm(0.01)}$	0.95$^{\pm(0.01)}$	**1.17**$^{\pm(0.09)}$	1.16$^{\pm(0.08)}$	0.97$^{\pm(0.16)}$	1.12$^{\pm(0.09)}$	**0.83**$^{\pm(0.00)}$	0.82$^{\pm(0.03)}$	**0.83**$^{\pm(0.00)}$	0.82$^{\pm(0.01)}$
FP_t	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.72$^{\pm(0.14)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.66$^{\pm(0.19)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.78$^{\pm(0.10)}$	**1.00**$^{\pm(0.00)}$
SR_t	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	**0.74**$^{\pm(0.05)}$	0.70$^{\pm(0.14)}$	0.68$^{\pm(0.09)}$	0.00$^{\pm(0.00)}$
Measures	G24_4				G24_5				G24_6a			
	Epsilon	Feasibility	Penalty	Stochastic	Epsilon	Feasibility	Penalty	Stochastic	Epsilon	Feasibility	Penalty	Stochastic
AE_t	NaN	NaN	NaN	NaN	124.47$^{\pm(67.67)}$	126.37$^{\pm(57.62)}$	148.83$^{\pm(42.22)}$	**23.37**$^{\pm(22.30)}$	793.57$^{\pm(47.74)}$	793.70$^{\pm(54.25)}$	747.50$^{\pm(95.12)}$	**556.90**$^{\pm(103.85)}$
CS_t	NaN	NaN	NaN	NaN	261.12$^{\pm(132.88)}$	263.26$^{\pm(112.28)}$	303.74$^{\pm(95.67)}$	**57.46**$^{\pm(44.34)}$	1214.64$^{\pm(398.88)}$	1253.21$^{\pm(434.65)}$	**942.23**$^{\pm(294.79)}$	1622.04$^{\pm(1157.78)}$
PR_t	**1.15**$^{\pm(0.06)}$	1.15$^{\pm(0.09)}$	1.01$^{\pm(0.12)}$	1.11$^{\pm(0.08)}$	**0.62**$^{\pm(0.02)}$	**0.62**$^{\pm(0.02)}$	0.60$^{\pm(0.06)}$	0.61$^{\pm(0.03)}$	0.91$^{\pm(0.00)}$	0.91$^{\pm(0.00)}$	0.85$^{\pm(0.11)}$	**0.92**$^{\pm(0.01)}$
FP_t	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.71$^{\pm(0.16)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.84$^{\pm(0.10)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.94$^{\pm(0.09)}$	**1.00**$^{\pm(0.00)}$
SR_t	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.48$^{\pm(0.04)}$	0.48$^{\pm(0.04)}$	**0.49**$^{\pm(0.03)}$	0.41$^{\pm(0.02)}$	0.65$^{\pm(0.14)}$	0.63$^{\pm(0.16)}$	**0.79**$^{\pm(0.12)}$	0.34$^{\pm(0.12)}$
Measures	G24_6b				G24_6c				G24_6d			
	Epsilon	Feasibility	Penalty	Stochastic	Epsilon	Feasibility	Penalty	Stochastic	Epsilon	Feasibility	Penalty	Stochastic
AE_t	730.27$^{\pm(62.03)}$	712.43$^{\pm(57.65)}$	748.07$^{\pm(112.38)}$	**506.27**$^{\pm(109.47)}$	717.53$^{\pm(49.96)}$	719.23$^{\pm(45.86)}$	789.23$^{\pm(147.43)}$	**505.27**$^{\pm(123.24)}$	646.50$^{\pm(45.52)}$	656.73$^{\pm(42.52)}$	626.10$^{\pm(40.45)}$	**421.33**$^{\pm(71.61)}$
CS_t	1023.74$^{\pm(152.18)}$	**984.93**$^{\pm(93.95)}$	1133.43$^{\pm(465.92)}$	1393.39$^{\pm(1035.71)}$	956.71$^{\pm(141.80)}$	**942.23**$^{\pm(175.47)}$	1143.82$^{\pm(497.61)}$	1471.65$^{\pm(1106.47)}$	**682.92**$^{\pm(72.61)}$	772.63$^{\pm(133.90)}$	688.02$^{\pm(99.53)}$	1181.31$^{\pm(647.70)}$
PR_t	0.91$^{\pm(0.00)}$	0.91$^{\pm(0.00)}$	0.83$^{\pm(0.12)}$	**0.92**$^{\pm(0.01)}$	0.91$^{\pm(0.00)}$	0.91$^{\pm(0.02)}$	0.88$^{\pm(0.05)}$	**0.92**$^{\pm(0.01)}$	1.41$^{\pm(0.07)}$	1.42$^{\pm(0.10)}$	**1.43**$^{\pm(0.10)}$	1.40$^{\pm(0.08)}$
FP_t	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.85$^{\pm(0.10)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.86$^{\pm(0.10)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$
SR_t	0.71$^{\pm(0.10)}$	**0.72**$^{\pm(0.09)}$	0.66$^{\pm(0.11)}$	0.36$^{\pm(0.18)}$	0.75$^{\pm(0.11)}$	**0.76**$^{\pm(0.12)}$	0.69$^{\pm(0.13)}$	0.34$^{\pm(0.15)}$	**0.95**$^{\pm(0.07)}$	0.85$^{\pm(0.13)}$	0.91$^{\pm(0.10)}$	0.36$^{\pm(0.14)}$
Measures	G24_7				G24_8b							
	Epsilon	Feasibility	Penalty	Stochastic	Epsilon	Feasibility	Penalty	Stochastic				
AE_t	NaN	NaN	NaN	NaN	242.37$^{\pm(404.83)}$	260.27$^{\pm(401.82)}$	348.40$^{\pm(441.93)}$	**84.70**$^{\pm(221.61)}$				
CS_t	NaN	NaN	NaN	NaN	9088.75$^{\pm(484.85)}$	7808.00$^{\pm(930.57)}$	7488.57$^{\pm(1471.50)}$	**6352.50**$^{\pm(497.94)}$				
PR_t	0.88$^{\pm(0.01)}$	**0.88**$^{\pm(0.00)}$	0.85$^{\pm(0.03)}$	0.88$^{\pm(0.01)}$	**0.94**$^{\pm(0.05)}$	**0.94**$^{\pm(0.05)}$	0.64$^{\pm(0.19)}$	0.77$^{\pm(0.07)}$				
FP_t	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.66$^{\pm(0.15)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	**1.00**$^{\pm(0.00)}$	0.57$^{\pm(0.24)}$	**1.00**$^{\pm(0.00)}$				
SR_t	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.00$^{\pm(0.00)}$	0.03$^{\pm(0.04)}$	0.03$^{\pm(0.05)}$	**0.14**$^{\pm(0.10)}$	0.01$^{\pm(0.03)}$				

TABLE 4.5: Average and standard deviation of offline error values obtained by all the repairs methods with $k = 0.5$, $S = 10$, 20 and 50, and $f_c = 1000$. Best results are remarked in boldface.

Algorithms	$S = 10$				
	G24_1	**G24_f**	**G24_2**	**G24_3**	**G24_3b**
Reference	0.07(±0.029)	0.029(±0.022)	0.394(±0.212)	0.041(±0.025)	0.058(±0.027)
Offspring	0.07(±0.053)	0.036(±0.036)	0.451(±0.317)	0.068(±0.056)	0.073(±0.048)
Mutant	0.271(±0.051)	0.095(±0.048)	0.29(±0.021)	0.159(±0.031)	0.193(±0.041)
Gradient	**0.043(±0.028)**	**0.004(±0.003)**	**0.259(±0.012)**	**0.01(±0.004)**	**0.033(±0.015)**
	G24_3f	**G24_4**	**G24_5**	**G24_7**	**G24_8b**
Reference	0.007(±0.004)	0.071(±0.035)	0.071(±0.024)	0.12(±0.088)	0.105(±0.062)
Offspring	0.04(±0.083)	0.067(±0.031)	0.089(±0.035)	0.253(±0.128)	0.114(±0.056)
Mutant	0.046(±0.019)	0.187(±0.045)	0.126(±0.021)	0.208(±0.034)	0.338(±0.048)
Gradient	**0.002(±0.003)**	**0.032(±0.013)**	**0.024(±0.007)**	**0.021(±0.008)**	**0.031(±0.009)**
Algorithms	$S = 20$				
	G24_1	**G24_f**	**G24_2**	**G24_3**	**G24_3b**
Reference	0.078(±0.042)	0.026(±0.019)	0.406(±0.328)	0.02(±0.009)	0.06(±0.036)
Offspring	0.086(±0.061)	0.03(±0.025)	0.416(±0.321)	0.039(±0.033)	0.035(±0.023)
Mutant	0.246(±0.047)	0.1(±0.05)	0.296(±0.02)	0.156(±0.033)	0.207(±0.031)
Gradient	**0.048(±0.026)**	**0.004(±0.004)**	**0.258(±0.009)**	**0.004(±0.002)**	**0.035(±0.017)**
	G24_3f	**G24_4**	**G24_5**	**G24_7**	**G24_8b**
Reference	0.008(±0.005)	0.06(±0.032)	0.075(±0.033)	0.107(±0.045)	0.108(±0.041)
Offspring	0.023(±0.029)	0.043(±0.038)	0.092(±0.048)	0.213(±0.075)	0.12(±0.069)
Mutant	0.05(±0.019)	0.218(±0.033)	0.132(±0.024)	0.267(±0.039)	0.333(±0.044)
Gradient	**0.002(±0.002)**	**0.033(±0.015)**	**0.029(±0.013)**	**0.021(±0.009)**	**0.033(±0.008)**
Algorithms	$S = 50$				
	G24_1	**G24_f**	**G24_2**	**G24_3**	**G24_3b**
Reference	0.069(±0.031)	0.031(±0.024)	0.371(±0.232)	0.011(±0.005)	0.045(±0.025)
Offspring	0.06(±0.032)	0.039(±0.037)	0.39(±0.187)	0.037(±0.069)	0.029(±0.025)
Mutant	0.26(±0.051)	0.1(±0.047)	0.298(±0.023)	0.1(±0.023)	0.161(±0.024)
Gradient	**0.043(±0.018)**	**0.003(±0.003)**	**0.257(±0.01)**	**0.002(±0.002)**	**0.027(±0.011)**
	G24_3f	**G24_4**	**G24_5**	**G24_7**	**G24_8b**
Reference	0.008(±0.009)	0.053(±0.042)	0.062(±0.018)	0.084(±0.024)	0.096(±0.041)
Offspring	0.03(±0.049)	0.038(±0.046)	0.08(±0.032)	0.2(±0.078)	0.111(±0.065)
Mutant	0.046(±0.018)	0.162(±0.022)	0.145(±0.025)	0.289(±0.037)	0.351(±0.04)
Gradient	**0.003(±0.003)**	**0.026(±0.011)**	**0.033(±0.012)**	**0.026(±0.011)**	**0.031(±0.007)**

4.3 Repair methods

In this section, through an empirical study, we investigate different repair methods to be applied in DCOPs. Among the other constraint handling techniques, repair methods has not only been suitable to deal with the constraints, but also has been able to improve the algorithm performance when has been used in dynamic environments. The reason is that this technique is not only choosing between the solutions in the selection, but also moves the solution towards feasible region by the repair operator. Indeed, the main idea of a repair method is to convert infeasible solutions into feasible ones. Based on the competitive results that these methods have shown, we carry out investigations on the behaviour of these repair methods for DCOPs.

TABLE 4.6: Statistical tests on the offline error values in Table 4.5. "X$^{(-)}$" means that the corresponding algorithm outperformed algorithm X. "X$^{(+)}$" means that the corresponding algorithm was dominated by algorithm X. If algorithm X does not appear in column Y means no significant differences between X and Y.

Functions	$S = 10$			
	Reference(1)	**Offspring(2)**	**Mutant(3)**	**Gradient(4)**
G24_1 (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$
G24_f (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_2 (44.2%)	$4^{(+)}$	$4^{(+)}$	$4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_3 (7.1-49.21%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_3b (7.1-49.21%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_3f (7.1%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_4 (0-44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_5 (0-44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_7 (0-44.2%)	$2^{(-)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(+)}$ and $4^{(+)}$	$1^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_8b (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
	$S = 20$			
	Reference(1)	**Offspring(2)**	**Mutant(3)**	**Gradient(4)**
G24_1 (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_f (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_2 (44.2%)	$4^{(+)}$		$4^{(+)}$	$1^{(-)}$ and $3^{(-)}$
G24_3 (7.1-49.21%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_3b (7.1-49.21%)	$2^{(+)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$
G24_3f (7.1%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_4 (4.75-44.2%)	$2^{(+)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$
G24_5 (4.75-44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_7 (4.75-44.2%)	$2^{(-)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(+)}$ and $4^{(+)}$	$1^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_8b (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
	$S = 50$			
	Reference(1)	**Offspring(2)**	**Mutant(3)**	**Gradient(4)**
G24_1 (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$
G24_f (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_2 (44.2%)	$3^{(+)}$		$1^{(+)}$ and $4^{(+)}$	$3^{(-)}$
G24_3 (7.1-18.63%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_3b (7.1-18.63%)	$2^{(+)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$
G24_3f (7.1%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_4 (28.9-44.2%)	$2^{(+)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$
G24_5 (28.9-44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_7 (28.9-44.2%)	$2^{(-)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_8b (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$

Based on the literature for the current repair methods applied in DCOPs, four types of repair methods including i) reference-based repair [87], ii) offspring-repair [94, 95], iii) mutant-repair [4, 6] and iv) gradient-based repair [18] have been distinguished. (i) uses reference solutions in order to convert an infeasible solution to a feasible one. In (ii) the repair method is similar to (i), the only difference between these two methods is that choosing the feasible reference solution in (i) is completely random, while in (ii) the nearest feasible reference solution is selected. (iii) is a repair method which does not require feasible solutions to operate, and is inspired by the differential mutation operator. (iv) is based on gradient information derived from the constraint set to systematically repair infeasible solutions.

Our main focus is to investigate the specifications of each of these methods on a recent benchmark set for DCOPs [87] when applying DE. For the comparison of the effectiveness of each method, the offline error [87] and two newly

proposed measures are used. The analysis shows that the gradient-based method outperforms other repair methods based on almost of the measures. However, this method can not be used like a black-box, since it should be known if the constraints have derivative. On the contrary, based on offline error, the worst method seem to be mutant repair method. But this method repairs the solutions very fast after only a few tries. Although, these small number of tries for repairing a solution in this method is mostly because in this benchmark, most of the problems have a huge feasible area. Finally, based on the analysis, the benefits and drawbacks of each method are pointed out and directions for future work are given.

The rest of this section is organized as follows. In Section 4.3.1, experimental setup is introduced. In Section 4.3.2, the experimental investigations regarding the effectiveness of repair methods with respect to different performance measures are described. The experimental results are divided in offline error analysis and success rate analysis and are presented in Section 4.3.2.1 and Section 4.3.2.2 respectively. Finally, in Section 4.3.3, we finish this chapter with some conclusions and directions for future work.

4.3.1 Experimental setup

The chosen benchmark problem originally has 18 functions [87] (see Chapter 2 for the benchmark), however in this work, 10 functions among them were used for the experiments. The reason for this selection was that part of these functions were not constrained and part of them did not have derivative for the constraints and could not be applied in gradient-based method. The test problems in this benchmark consist a variety of characteristics like i) disconnected feasible regions (1-3), ii) the global optima at the constraints' boundary or switchable between disconnected regions, or iii) the different shape and percentage of feasible area.

In the experiments, for the objective function, only medium severity is considered ($k = 0.5$), while different change severities are considered for the constraints ($S = 10$, 20 and 50). Based on the definition of the constrains in this benchmark [87], $S = 10$ represents for large severity, $S = 20$ for medium severity and $S = 50$ for the small severity of changes on the constraints. The frequency of change (f_c) is considered equal to 1000 evaluations (only in the objective function). Worth to mention that, in the repair methods, the constraints evaluations are not considered as extra evaluations when using for DCOPs [6]. More details on the benchmark can be found in [87].

The configurations for the experiments are as follows. The number of runs in the experiments are 50, and number of considered times for dynamic perspective of the test algorithm is $5/k$ ($k = 0.5$). Parameters relating to DDE algorithm are as follows: DE variant is DE/rand/1/bin, population size is 20, scaling factor (F) is a random number $\in [0.2, 0.8]$, and crossover probability is 0.2. In the experiments, four repair methods including Reference-based, Offspring, Mutant and Gradient-based, explained in Chapter 3, have been applied for handling the constraint in DE algorithm.

4.3.2 Experimental results

The experimental results are divided as i) offline error analysis and ii) success rate and required number of iterations. In these experiments, we investigate the behaviour of different repair methods in DE algorithm based on the previous defined measures. In the analysis, the effects of different severities on the constraints are considered for these ten test problems. We do not bring the results for changes of frequency since it does not have any effect in the behaviour of the repair methods.

4.3.2.1 Offline error analysis

The results obtained for the four repair methods using offline error are summarized in Table 4.5. Furthermore, for the statistical validation, the 95%-confidence Kruskal-Wallis (KW) test and the Bergmann-Hommels post-hoc test, as suggested in [37], are presented (see Table 4.6). Non-parametric tests were adopted because the samples of runs did not fit to a normal distribution based on the Kolmogorov-Smirnov test. Based on the results, for the constraint's change severity $S = 10$, the gradient-based repair outperformed almost all of the other methods in nine test problems (G24_f, G24_2, G24_3, G24_3b, G24_3f, G24_4, G24_5, G24_7 and G24_8b) except one test problem (G24_1) that in which offspring-repair has similar performance. For this severity, reference-based repair and offspring-repair performed almost the same for nine test problems (G24_1, G24_f, G24_2, G24_3, G24_3b, G24_3f, G24_4, G24_5 and G24_8b) except one test problem (G24_7), where reference-based repair outperformed offspring-repair. As Table 4.6 illustrates, mutant-repair is the worst among all the methods for eight test problems (G24_1, G24_f, G24_3, G24_3b, G24_3f, G24_4, G24_5 and G24_8b) except two test problems, in which has similar results with reference-based repair (G24_2) and offspring-repair (G24_2 and G24_7).

For the constraint's change severity $S = 20$, the gradient-repair excelled almost all the other methods in seven test problems (G24_1, G24_f, G24_3, G24_3f, G24_5, G24_7 and G24_8b) with exceptions including G24_2, G24_3b and G24_4, that in which offspring-repair had similar performance. For this change severity, reference-based repair and offspring-repair performed almost the same for seven test problems (G24_1, G24_f, G24_2, G24_3, G24_3f, G24_5 and G24_8b) except three test problems (G24_3b, G24_4 and G24_7). For these three problems, while in two test problems (G24_3b and G24_4) offspring-repair had better results, in one test problem (G24_7) reference-based repair outperformed the offspring-repair. Mutant-repair had the worst results between all the methods for eight test problems (G24_1, G24_f, G24_3, G24_3b, G24_3f, G24_4, G24_5 and G24_8b) except two test problems in which had similar results with reference-based repair (G24_2) and offspring-repair (G24_2 and G24_7).

For the constraint's change severity $S = 50$, the gradient-repair excelled the other methods in six test problems (G24_f, G24_3, G24_3f, G24_5, G24_7 and G24_8b) with exceptions of having similar performance with offspring-repair (G24_1, G24_2, G24_3b and G24_4) and reference-based repair (G24_2). For this change severity, reference-based repair and offspring-repair performed almost the same for seven test problems (G24_1, G24_f, G24_2, G24_3, G24_3f, G24_5 and G24_8b) except three test problems (G24_3b, G24_4 and G24_7). For these three problems, while in two test problems (G24_3b and G24_4) offspring-repair had better results, in one test problem (G24_7) reference-based repair outperformed the offspring-repair. Mutant-repair had the worst results between all the methods for nine test problems (G24_1, G24_f, G24_3, G24_3b, G24_3f, G24_4, G24_5, G24_7 and G24_8b) except one test problem (G24_2) in which showed similar results with offspring-repair.

Based on offline error, gradient-repair outperformed other methods for all severities. One reason is that in this work all test problems have the global optimum on the boundary of the constraints. Unlike gradient-repair, other methods take larger random steps towards feasible area so they often cross the boundary and as a result lose to reach the global optima. Although, gradient-repair cannot be applied for the functions that do not have derivative for their constraints. For this reason, the four functions G24_6a, G24_6b, G24_6c and G24_6d (that are functions inside this set of benchmark) were not used in our experiments. Therefore, for this method an understanding about the behaviour of the constraints is specifically needed. Changes in severity do not decrease the performance of this method. Even though, we observe that in severity $S = 50$ it outperformed other methods in less test problems. It is

TABLE 4.7: Average and standard deviation of: i)Success rate(s_r), ii) required number of iterations(rn_i) for each of the repairs methods with $k = 0.5$, $S = 10$, 20 and 50, and $f_c = 1000$. Best results are remarked in boldface.

Functions	Success rate(s_r)				Required number of iterations(rn_i)			
	Reference	**Offspring**	**Mutant**	**Gradient**	**Reference**	**Offspring**	**Mutant**	**Gradient**
	$S = 10$							
G24_1	99.95(±0.08)	99.94(±0.08)	**100.00(±0.00)**	99.72(±0.15)	67.98(±7.99)	79.78(±4.33)	**2.26(±0.03)**	4.30(±0.24)
G24_f	99.97(±0.05)	99.97(±0.05)	**100.00(±0.00)**	99.71(±0.17)	64.77(±8.41)	74.95(±8.33)	**2.26(±0.03)**	3.96(±0.23)
G24_2	99.99(±0.02)	99.96(±0.08)	**100.00(±0.00)**	99.78(±0.18)	51.20(±10.65)	70.18(±5.94)	**2.26(±0.04)**	3.80(±0.26)
G24_3	99.98(±0.04)	99.96(±0.06)	**99.98(±0.02)**	95.19(±1.73)	60.35(±7.92)	70.91(±6.07)	**4.74(±0.13)**	8.03(±1.64)
G24_3b	99.97(±0.07)	99.97(±0.06)	**99.99(±0.03)**	95.96(±1.38)	66.51(±7.95)	71.81(±7.50)	**4.74(±0.11)**	7.60(±1.30)
G24_3f	99.88(±0.11)	99.85(±0.12)	**99.94(±0.05)**	93.33(±2.21)	84.66(±3.47)	89.90(±3.49)	14.04(±0.25)	**10.26(±2.10)**
G24_4	99.98(±0.04)	99.94(±0.07)	**99.99(±0.02)**	95.45(±1.75)	62.23(±9.58)	72.54(±6.27)	**4.77(±0.15)**	8.05(±1.66)
G24_5	**95.76(±3.55)**	74.16(±11.24)	71.44(±0.69)	74.31(±1.90)	47.28(±6.52)	72.00(±3.66)	38.09(±0.74)	**28.79(±1.78)**
G24_7	**92.79(±6.18)**	75.81(±9.57)	72.40(±0.66)	75.55(±2.06)	63.57(±6.47)	73.20(±4.76)	37.30(±0.73)	**27.44(±1.97)**
G24_8b	99.97(±0.06)	99.95(±0.06)	**100.00(±0.00)**	99.75(±0.13)	63.29(±6.01)	69.54(±6.26)	**2.26(±0.03)**	3.94(±0.19)
	$S = 20$							
G24_1	99.97(±0.05)	99.94(±0.08)	**100.00(±0.00)**	99.75(±0.11)	66.87(±7.25)	78.58(±5.85)	**2.25(±0.04)**	4.22(±0.24)
G24_f	99.98(±0.04)	99.97(±0.07)	**100.00(±0.00)**	99.67(±0.19)	65.19(±8.69)	76.88(±6.91)	**2.26(±0.04)**	4.02(±0.26)
G24_2	99.97(±0.05)	99.95(±0.10)	**100.00(±0.00)**	99.83(±0.11)	50.61(±10.83)	69.63(±6.42)	**2.26(±0.03)**	3.75(±0.22)
G24_3	99.96(±0.07)	99.93(±0.09)	**100.00(±0.01)**	92.37(±2.48)	69.44(±5.67)	74.51(±4.81)	**6.11(±0.14)**	11.09(±2.37)
G24_3b	99.96(±0.07)	99.89(±0.11)	**100.00(±0.01)**	92.12(±2.83)	71.79(±6.88)	79.61(±3.70)	**6.08(±0.14)**	11.60(±2.68)
G24_3f	99.90(±0.11)	99.84(±0.13)	**99.94(±0.04)**	92.96(±2.17)	85.00(±3.61)	90.97(±2.55)	14.12(±0.23)	**10.63(±2.04)**
G24_4	99.96(±0.07)	99.89(±0.10)	**99.99(±0.02)**	92.54(±2.23)	72.50(±6.14)	79.75(±3.84)	**6.12(±0.12)**	11.22(±2.11)
G24_5	97.47(±1.19)	82.19(±9.70)	**100.00(±0.00)**	92.46(±2.64)	39.24(±11.07)	69.08(±4.56)	**4.96(±0.11)**	10.85(±2.53)
G24_7	96.09(±1.84)	86.39(±5.93)	**100.00(±0.00)**	93.03(±2.25)	56.24(±6.08)	69.03(±4.75)	**4.90(±0.11)**	10.41(±2.17)
G24_8b	99.97(±0.05)	99.95(±0.07)	**100.00(±0.00)**	99.70(±0.20)	61.55(±5.67)	68.86(±7.18)	**2.27(±0.04)**	3.95(±0.26)
	$S = 50$							
G24_1	99.97(±0.05)	99.91(±0.10)	**100.00(±0.00)**	99.70(±0.15)	68.09(±8.06)	78.75(±4.43)	**2.26(±0.04)**	4.28(±0.23)
G24_f	99.97(±0.05)	99.95(±0.07)	**100.00(±0.00)**	99.69(±0.15)	64.98(±10.06)	74.94(±7.47)	**2.26(±0.04)**	4.02(±0.27)
G24_2	99.99(±0.02)	99.93(±0.11)	**100.00(±0.00)**	99.84(±0.08)	50.01(±10.05)	69.64(±5.59)	**2.27(±0.04)**	3.76(±0.17)
G24_3	99.92(±0.09)	99.89(±0.13)	**99.99(±0.01)**	92.90(±2.42)	75.99(±4.35)	80.88(±4.04)	**8.63(±0.15)**	10.62(±2.31)
G24_3b	99.92(±0.10)	99.86(±0.12)	**99.99(±0.02)**	93.04(±2.19)	77.14(±5.02)	83.60(±2.87)	**8.63(±0.18)**	10.82(±2.12)
G24_3f	99.91(±0.09)	99.84(±0.11)	**99.95(±0.04)**	93.18(±2.16)	84.12(±3.39)	90.11(±3.96)	14.01(±0.25)	**10.40(±2.08)**
G24_4	99.90(±0.10)	99.85(±0.13)	**99.99(±0.02)**	93.36(±2.01)	78.17(±4.62)	84.28(±2.89)	**8.66(±0.15)**	10.51(±1.91)
G24_5	97.64(±1.30)	86.30(±8.46)	**100.00(±0.00)**	91.57(±3.13)	40.28(±9.01)	66.52(±6.61)	**2.83(±0.05)**	11.66(±2.99)
G24_7	96.88(±1.65)	89.71(±6.01)	**100.00(±0.00)**	93.04(±2.82)	55.12(±6.69)	66.17(±5.68)	**2.82(±0.04)**	10.38(±2.70)
G24_8b	99.97(±0.06)	99.97(±0.07)	**100.00(±0.00)**	99.77(±0.16)	63.97(±7.20)	71.18(±6.15)	**2.27(±0.03)**	3.89(±0.24)

because offspring-repair performance increased for some test problems for this severity. Similar behaviour in reference-based repair and offspring-repair based on offline error for all the severities is due to the similar procedure (uniform crossover in GA) that they use for repairing the infeasible solutions. The only difference is the way they choose the reference solution.

4.3.2.2 Analysis of success rate and required number of iterations for repairing solutions

Regardless of severity, the total number of infeasible solutions (n_T) that needed repair for different functions were in the range between 1882 and 2981. The n_T values were increased for the functions that had dynamic constraints like G24_3, G24_3b, G24_4, G24_5 and G24_7. The reason is because, when the constraints are changing, it is more probable that some feasible solutions be

TABLE 4.8: Main features of each repair method

Method	Advantages	Disadvantages
Reference	i) Maintain infeasible solution information, ii) increase diversity	i) Random behaviour ii) high number of required iterations and iii) reference solutions needed
Offspring	i) Maintain infeasible solution information	i) High number of required iterations and ii) roughly random behaviour
Mutant	i) High success rate, ii) low iterations needed, iii) no reference solution needed, iv) increase diversity	i) Not a good performance (offline error) ii) loose the information and iii) random behaviour
Gradient	i) Prominent performance when the optimal solution is in the boundaries of the feasible area, ii) good performance (offline error), iii) no reference solution needed, iv) maintain infeasible solution information and v) low iterations needed	i) Knowledge about the characteristic of constraints needed and ii) only can be applied when the constraints have derivate

converted to infeasible ones after a change occurs.
The results for the success rate (s_r) and required number of iterations (nr_i) measures are presented in Table 4.7. Regarding to these results, some general observations can be concluded. The number of required iterations (nr_i), was the smallest for mutant-repair with a range between 2 to 8 and in second place is gradient-repair with the range between 4 to 10. An exception of this trend was seen in the function G24_3f in all the severities, and functions G24_5 and G24_7 for the severity $S = 10$, which gradient-repair excelled mutant-repair since the percentage of feasible area in these cases were small (see Table 4.6 for the feasibility percentages). Overall, in mutant-repair method since the process of producing a feasible solution is completely random and in this applied benchmark functions, the percentage of feasible area is huge, so this method achieved to feasible solutions after a few tries. In another words, this method is roughly dependent to the percentage of the feasible area. As mentioned before, the second smallest values for this measure was for gradient-based method; but in this case the reason was based on this method's wise selection and the fact that it only moves in the direction and with the amount of satisfying the constraint violations.

The worst results for this measure belonged to offspring-repair with an average number of required iterations ranging from 66 to 91 and reference-based repair with a range from 47 to 85. Compared to offspring-repair, reference-based repair required lower number of iterations, and this was because offspring-repair's step sizes are smaller and for this reason it needed more iterations to convert the infeasible solution to a feasible one. Other drawback in these two methods is that a number of evaluations is needed to produce feasible

reference population. This can be expensive in high computational complex problems [40].

Generally, based on considering all the measures, offspring and reference-based repair methods in most functions had similar behaviours. This is mostly because, the process of converting the infeasible solutions to feasible ones are approximately the same in these methods, and the only difference is the way they choose the feasible reference solution. They do not loose the information of infeasible solution completely, as they use this individual to move in the direction of one of the feasible solutions (this is more evident in offspring as it uses the nearest reference feasible solution).

As regards to the third measure (success rate), although mutant-repair has the best values, but this is because in this set of benchmark, most of the test problems has a huge feasible area. For this reason, reaching to a feasible solution randomly after a few tries is easily possible based on this method. Obviously, for the cases of small feasible area, this method's efficiency will decrease. This was the case for the functions G24_5 and G24_7; as can be seen from Table 4.5, the values for this measure dropped drastically for this method as the percentage of feasible area is small for some time periods in these two functions. Reference-based and offspring were on the second place based on the values of this measure and the results of these two methods are roughly similar. Although, gradient-based method seemed to have worse results based on this measure, the differences between these values and the values for other methods were not significant. Moreover, practically, there is no need to convert all the infeasible solutions. In Table 4.8, a review of the advantages and disadvantages of each method is presented.

4.3.3 Conclusions and discussions

In this section, an investigation on different current repair methods in DCOPs were carried out. For the comparison, three different measures including offline error, success rate and the average number of required iterations for repairing the infeasible individuals were used. The results showed regardless of the change severities, in most cases gradient-based method outperformed the other methods based on offline error. This method especially performs much better than the other methods for the problems that have the optimal solution in the boundaries of the feasible area. Indeed, this method moves in very small steps and will not lose the optimal solution in the boundaries. Although, this method can not be applied for the functions that do not have derivative of the constraints. For the other measurement criteria, the number

of required repair for mutant repair was the smallest and the second rank was for gradient-based method. Finally, based on the success rate, all of the repair methods were able to repair most of the infeasible solutions. Such promising performance was based on the fact that the feasible region of the main static test problem ((G24) [65]) occupy around 79% of the whole search space [4]. For future work, a combination of different repair methods can be investigated in order to make the most of each method.

Chapter 5

A Benchmark Generator for Dynamic Constrained Continuous Problems

5.1 Introduction

Besides developing algorithms, there should be a comprehensive benchmark that can test algorithms considering a range of characteristics. Although, there are a range of benchmarks proposed to test the relevant algorithms for discrete spaces [105], and/or multi-objective optimisation in dynamic environments [59], for continuous spaces in single objective optimisation so far, the most used benchmark is the proposed benchmark by Nguyen [86]. In this benchmark, the dynamic changes are applied by adding time-dependent terms to the objective function and the constraints of one of the functions (G_24) of the static benchmark proposed in CEC 2006 [66].

Although, there are parameters included in Nguyen benchmark to alter the severity of environmental changes, this benchmark is based on only one objective function and the transformation of this function. Thus it is not applicable to test different characteristics. Moreover, all of the problems in this benchmark are two-dimensional and are not extensible to larger problem dimensions. In addition, the feasible regions of the dynamic constraint function in this benchmark are very large, which might not be sufficiently complicated [19]. Bu et al in [19], introduces one variant of this benchmark that has a parameter to control the size and the number of the feasible regions. The other variant introduced in this paper is based on the moving peak benchmark.

The proposed benchmark in [135] is based on dynamic transformations of Nguyen benchmark [86]. However, the problem information including the number of feasible regions, the global optimum, and the dynamics of each

feasible region is lacking. The lack of such information makes it difficult to measure and analyze the performance of an algorithm and probably this is the reason this benchmark have become less popular than Nguyen benchmark [86].

In terms of having a scalable and flexible benchmark, there are some benchmark generators proposed in the literature. Like in [63], a dynamic benchmark generator is proposed that is designed with the idea of constructing dynamic environments across binary, real, and combinatorial solution spaces. The dynamism is obtained by tuning some system control parameters, creating six change types including small step, large step, random, chaotic, recurrent, and recurrent change with noise.

While the aforementioned benchmark generator's main focus is on creating dynamic objective functions, we concentrate on creating dynamism in the constraints in this study. Our motivation comes from characteristic of some real-world problems such as scheduling power system problem having dynamic linear constraints (due to the variable demand and available resources over-time). For a better insight about the effects of constraint changes, we keep the objective function static. Indeed, this is the case in some real world problems in which only constraints will change such as the problem of hydro-thermal power scheduling in continuous spaces [36] or the ship scheduling problem in discrete spaces [75].

Dynamic changes are imposed by the translation and rotation of the constraint's hyperplane. The examples of these two operations on the constraint in a real-world dynamic environment are: the reduction and increment of demand that happens regularly at power system (hyperplane translation) or changes on the share of each power production plant (hyperplane rotation) [81]. Our proposed benchmark generator is flexible (frequency and severity of changes, number of environmental changes, and dimension of the problem), simple to implement (with any objective function), analyze, or evaluate and computationally efficient and finally allows conjectures to real-world problems.

In the experiments, we apply DE algorithm with different constraint handling techniques and observe how they deal with these changes depending on the magnitude and frequency of changes. Our experiments are repeated across some well-known functions including sphere, Rastrigin, Ackley and Rosenbrock. For the analysis on the performance of the tested algorithms, a ranking procedure is introduced that uses the values of the objective function and the constraint violations to rank the performance of the algorithms. In addition,

the common modified offline error is also evaluated for the experiments and the results are investigated. The results reveal that the changes on frequency and hyperplane rotation and translation have a direct correlation with the performance of the constraint handling techniques. Therefore, we could effectively put the algorithms to struggle and test their performance with imposing simple linear changes.

The remainder of this chapter is as follows. In Section 5.2, our proposed dynamic changes' framework is described. Experimental setup is presented in Section 5.3 followed by experimental results in Section 5.4. Finally in Section 5.5 conclusions and future work are summarized.

5.2 Dynamic changes framework

Many of the real-world problems have single or multiple linear constraints. Therefore, the relevant benchmark can be as simple as creating some changes in the constraint's coefficients and boundaries that represent changes in different times. In this section, we will introduce a framework to create changes on linear constraints to emulate a dynamic environment. Our proposed changes is observed in some real-world problems such as power scheduling problem [81]. In this problem, the conditions in the system such as demand or available resources will change over time. In the remaining of this section, first the constraint setup is presented and then the frequency setup will be explained.

5.2.1 Constraint setup

For emulating the dynamic constraints, simple linear constraints are used for search space modification. Although linear constraints are simple, they are used as a representation of some of the real world problem constraints and prevents over-complication of the analysis. The general formulation for the linear constraints are as follows.

$$g_i(\vec{x}) = \sum_{j=1}^{D} a_j x_j - b_i \leq 0 \quad i \in \{1, \ldots, m\} \tag{5.1}$$

where $g_i(\vec{x})$ is the ith inequality constraint, a_j is the jth variable coefficient, x_j is the jth decision variable, b_i is the upper limit of the ith constraint and D is the problem dimension. First, a general case for one constraint is defined and then is developed for multiple constraints.

Two operations are defined for changes on constraint, the first one is related to changes on b coefficient (hyperplane translation) and the second one is changes of a_i coefficients (hyperplane rotation). These two changes in the simple linear constraint can happen commonly in real-world problems. Example of hyperplane translation can be the changes of demand or capacity of each production plants including stochastic renewable plants in a scheduling power plant problem. Similarly example of hyperplane rotation can be the changes on the share of each plant to produce overall supply, or the probability of renewable resources production.

Hyperplane translation: If a_i coefficients are chosen in a way to create a unit normal vector of $\vec{a}$, changes of b will directly show the effects of changing the distance from the optimum point[1]. The distance (d) of the constraint hyperplane from the origin 0^D is defined in Equation 5.2.

$$d = b / \left(\sum_{i=1}^{D} a_i^2 \right)^{1/2} \tag{5.2}$$

The constraint bound value ($b(t)$) at time t is obtained by adding a random value to its previous time value ($b(t-1)$) as in Equation 5.3.

$$b(t) = b(t-1) + k_r \tag{5.3}$$

where k_r is chosen uniformly at random within the interval $[lk, uk]$. Figure 5.1 shows an example of these settings for creating different magnitudes of change for single constraint case. These values are only samples of constraints boundaries for creating dynamic environment. As mentioned before, we can create multiple benchmarks for testing the algorithms with different scales of the changes based on the problem type. The two criteria that will affect the proper choice of the values of lk and uk are the dimension of the problem and the variable ranges.

Hyperplane rotation: In this operation, the changes are created by rotating the hyperplane at each separate time. Random numbers are created in $\in [0, 1]$ such that we have a unit normal vector of $\vec{a}$. For any time we randomly select some of the coefficients and swap their values. As in this way, $\vec{a}$ is still a unit normal vector, therefore the changes at each time is only related to the rotation and not the translation of hyperplane. So by making these changes at each time, we will have a rotated hyperplane and we can observe how it will effect the algorithm's behaviour.

[1] for all the chosen functions zero is the optimum point

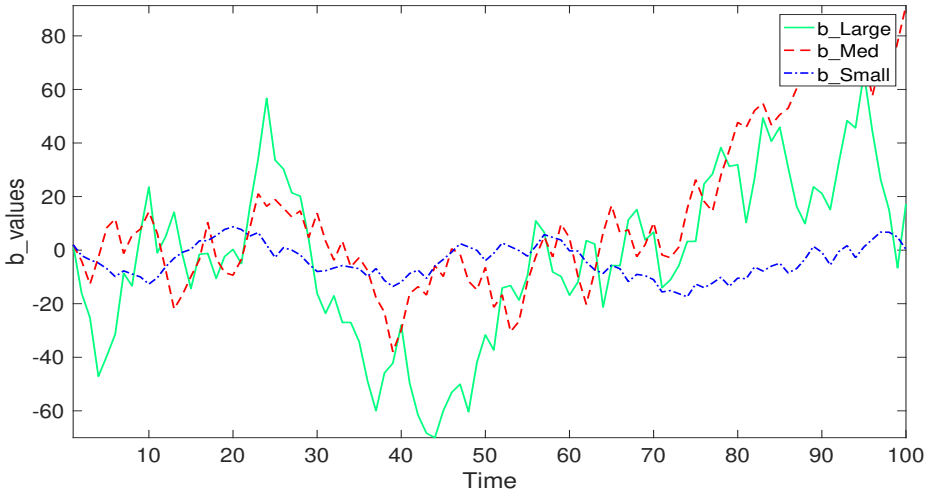


FIGURE 5.1: Sample settings for large, medium and small changes on b_values

In another setting, both changes of a_i and b values are considered. In this case, we have changes on either b or a_i based on a known probability at any separate time.

With the current settings of the changes on the linear constraint coefficient, we can observe how the algorithms response to the new modified search space. With the comparisons, we will observe which one of the compared algorithms will react faster in order to converge to the optimum after a change occurs. The coefficients of the linear constraints are generated through a proposed constraint generator and are then fed to the algorithm. Since the constraint coefficients are random, for a fair algorithms comparison, it is needed that all the generated coefficients for each run be the same for all the algorithms and a per algorithm generation would lead to a difference in compared constraints. At each environment change, the number of feasible and infeasible solutions will change so that we expect to observe how efficiently the algorithms will manage the new set of constraints within the problem.

In many real-world problems including hydro-thermal scheduling problem, multiple constraints rather than single constraint will define the dynamic environment of the problem. In this case, changes are imposed on b_i value of ith constraint. In order to avoid complexity, at each time we only change one of the constraints boundaries. This is aligned with the real-problems that one or a few criteria and not all will change at the new environment condition.

5.2.2 Frequency setup

The frequency of change (τ) is defined as how often the problem changes. A higher frequency seems to be more difficult for an algorithm to solve the

related problem as less time is available at each period to reach the new global optimum [104]. In the literature of DCOPs, the number of fitness evaluations is considered as a criteria to represent how frequently a change occurs [87].

Three different constraint handling techniques, introduced in Chapter 3, including penalty [116], feasibility rules [35] and ϵ-constrained [114] are chosen to be included as for handling constraint with DE algorithm. With different constraint handling techniques, we will observe how these algorithms will respond to the new changes in the environment.

5.3 Experimental setup

The experiments are conducted for different magnitudes of hyperplane translation (first experiment), changes of frequency (second experiment), and a combination of hyperplane translation and rotation (third experiment). The settings for b_values for hyperplane translation are large: lk=-25, uk=25, medium: lk=-15, and uk=15 and small: lk=-5, uk=5 with initial value of b: b_0=2. The settings for changes of frequency are large ($\tau = 500$), medium ($\tau = 1000$) and small ($\tau = 2000$). The maximum number of evaluations is obtained by: $100\tau + 1000$, where 100 is the number of changes in the environments and 1000 is the buffer that allows the algorithms proceed in their optimisation process for the first time before a change occurs. The first column of the tables 5.1 and 5.3 show severities on magnitude of hyperplane translation, and the second column show severities of frequency.

The results are repeated for four artificial functions including sphere, Rastrigin, Ackley and Rosenbrock. Parameters of DE are chosen as $NP = 20$, $CR = 0.2$ and F is a random number in $\in [0.2, 0.8]$. The dimension of the problem is 30 for all the experiments.

Two distinct measurements are applied for comparing the algorithms, the first one is the modified offline error [87] explained in chapter 2 and the second one is the following ranking procedure.

5.3.1 Ranking mechanism

The following ranking procedure is build upon the feasibility rules [35]. Considering the best solution obtained before a change in time for algorithms i and j, a lexicographical ordering mechanism in which the minimization of

the sum of constraint violation precedes the minimization of the objective function will define the ranking of the algorithms (Equation 5.4).

$$\begin{aligned} &(f(\vec{x}_{Ai,t}), \phi(\vec{x}_{Ai,t})) < (f(\vec{x}_{Aj,t}), \phi(\vec{x}_{Aj,t})) \Leftrightarrow \\ &\begin{cases} f(\vec{x}_{Ai,t}) < f(\vec{x}_{Aj,t}), & \text{if } \phi(\vec{x}_{Ai,t}) = \phi(\vec{x}_{Aj,t}) \\ \phi(\vec{x}_{Ai,t}) < \phi(\vec{x}_{Aj,t}), & \text{otherwise} \end{cases} \end{aligned} \tag{5.4}$$

where $f(\vec{x}_{Ai,t})$ and $\phi(\vec{x}_{Ai,t})$, are the objective function and the sum of constraint violation (Equation 5.5) of the best solution achieved with algorithm i before a change happens respectively. The sum of constraint violation $\phi(\vec{x}, t)$ is calculated as follows:

$$\phi(\vec{x}, t) = \sum_{i=1}^{m} max(0, g_i(\vec{x}, t)) + \sum_{j=1}^{p} |h_j(\vec{x}, t)| \tag{5.5}$$

where the inequality ($g_i(\vec{x}, t)$) and equality ($h_j(\vec{x}, t)$) constraints have been defined in chapter 2.

This lexicographical ranking procedure is applied across every test conducted for analytical testing. In each time change, the performance of the algorithms are ranked and these scores are combined into an overall performance score. Once the overall scores are calculated, the algorithms are ranked in order of the performance. We applied this ranking procedure for three algorithms, although it is adaptable to be used for comparing any number of algorithms (multi-compare).

5.4 Experimental results

In this section, a complementary method to compare the algorithms qualitatively is followed by the statistical test results which are divided to two cases: i) single constraint and ii) multiple constraints.

5.4.1 Illustration of results for sphere

One qualitative way that helps compliment the comparison of the algorithms is to plot the objective values and sum of constraint violation for different changes in the environment, Figure 5.2. This figure will represent the differences between the objective function values of each algorithm (averaged across thirty runs) for the last generation before a change in time. The dot

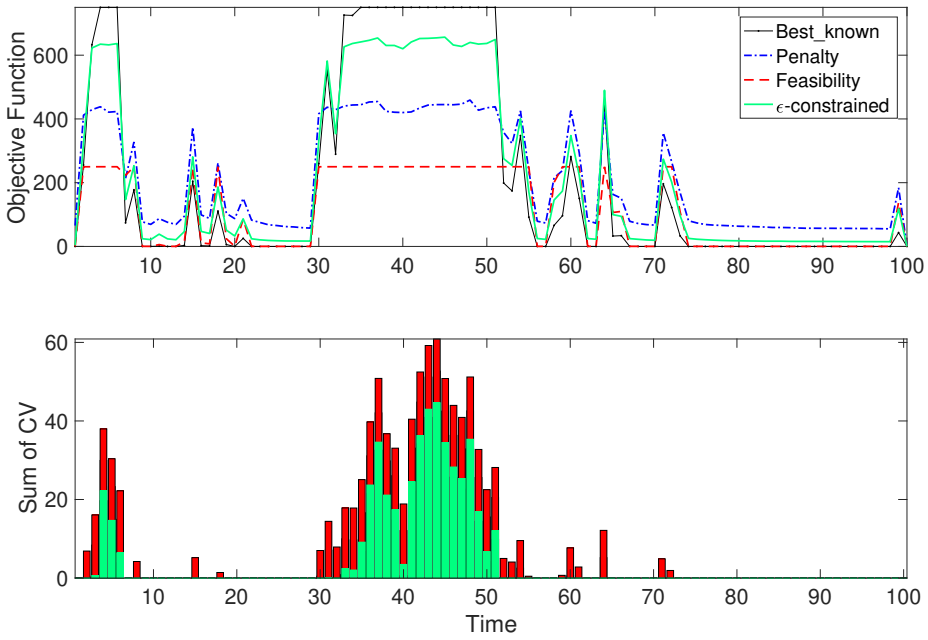


FIGURE 5.2: Sphere objective function and sum of constraint violation over-time

coded plot shows the best-known solution. In the top graph, the y-axis represents the values of the objective function and in the bottom graph a bar chart is representing the sum of constraint violations for the corresponding time for each algorithm. This figure belongs to the relevant details of the sphere function with medium frequency and large amplitude of hyperplane translation changes. As the figure shows, the created changes by the hyperplane translation, have successfully created the new environment and made the algorithms to struggle to find the new optimum. At first look it seems feasibility and ϵ-constrained are more successful to reach near-optimum solutions, however the bottom graph shows for some of the times they reach to infeasible solutions with the shown sum of constraint violation. Thus, for a better comparison of algorithms, the statistical analysis are essential.

In addition to the figure, Table 5.1 shows part of the results (25 times out of 100) of applying the benchmark for testing the sphere function with a single dynamic linear constraint. As the table shows, the hyperplane translation (medium changes of b_values) changes the size of the feasible region[2] inside of the search space. In this case there is an inverse relation between the size of the feasible region for the sphere function and b_value that is observable in the results. As the feasible region changes over time, new optimal points can appear, represented in the table as best_known. The way the algorithms track

[2]The feasible region is calculated by generating one million random solutions and getting the percentage of those that do not have constraint violations.

TABLE 5.1: Testing benchmark for single constraint setup (sphere function)

Time	b_Values	Feasible region(%)	Best-Known	Penalty	Feasibility	ϵ-constrained
25	18.90	100	0	68.24(±8.56)	0.14(±0.06)	19.76(±3.23)
26	15.68	100	0	65.68(±8.56)	0.13(±0.04)	18.29(±3.46)
27	12.36	99.99	0	63.22(±8.30)	0.13(±0.04)	18.1554(±3.35)
28	14.68	100	0	61.82(±7.02)	0.12(±0.04)	18.0836(±3.24)
29	4.72	94.90	0	61.73(±6.98)	0.11(±0.03)	17.65(±3.07)
30	13.81	100	0	60.67(±6.92)	0.10(±0.02)	17.40(±3.02)
31	3.43	88.29	0	61.16(±8.46)	0.10(±0.02)	17.21(±2.72)
32	-3.73	9.88	13.86	120.29(±19.31)	50.79(±3.04)	72.16(±10.62)
33	3.36	87.71	0	78.24(±12.88)	0.23(±0.10)	23.34(±5.39)
34	-6.24	1.52	38.83	186.09(±30.30)	124.59(±2.69)	105.79(±13.04)
35	-2.79	16.65	7.79	103.78(±14.56)	31.22(±2.56)	57.34(±8.82)
36	-7.59	0.40	57.416	208.18(±45.91)	176.86(±1.20)	129.04(±12.59)
37	-17.71	0	312.29	432.27(±43.40)*	249.90(±0.26)*	373.68(±8.93)*
38	-23.18	0	535.18	449.60(±40.75)*	249.96(±0.19)*	564.60(±8.40)*
39	-37.95	0	750	445.62(±37.58)*	249.99(±0.02)*	622.19(±26.84)*
40	-29.763	0	750	446.50(±39.38)*	249.95(±0.17)*	631.42(±27.00)*
41	-16.24	0	262.70	406.37(±46.45)*	249.94(±0.15)*	332.44(±9.65)*
42	-13.63	0	184.99	346.09(±31.43)	249.93(±0.16)*	262.38(±11.48)*
43	-16.74	0	279.02	416.34(±37.78)*	249.98(±0.05)*	345.43(±9.60)*
44	-5.81	2.18	33.62	160.23(±23.39)	109.16(±2.41)	98.47(±11.50)
45	-9.827	0.02	96.12	248.92(±28.16)	249.73(±1.24)*	175.17(±9.74)
46	0.51	57.04	0	86.80(±12.70)	0.37(±0.16)	31.98(±6.47)
47	-1.52	29.99	2.29	94.192(±18.16)	11.24(±2.03)	43.25(±12.46)
48	-11.59	0.00	133.88	282.30(±33.81)	249.86(±0.30)*	211.62(±11.46)
49	-14.86	0	219.82	378.82(±44.47)*	249.98(±0.07)*	292.40(±14.35)*
50	-6.50	1.18	42.07	177.18(±26.29)	134.64(±2.25)	111.69(±10.12)

these new best-known solutions is recorded across all of the runs and then the distribution is measured in the table. This allows a comparison between algorithms for specific times, although, for larger time scales individual comparisons are not preferable. The highlighted points with an asterisk shows that the relevant algorithm has sum of constraint violation other than zero for the relevant times.

5.4.2 Single constraint

The results of this section and the next section (multiple constraints) are based on measuring the performance of the algorithms with the ranking mechanism and the modified offline error explained in Chapter 2; where the results are summarized in Tables 5.2 and 5.3. To validate the results, the 95%-confidence Kruskal-Wallis statistical test and the Bonferroni post hoc test, as suggested in [37], are presented. Nonparametric tests were adopted because the samples of runs did not fit to a normal distribution based on the Kolmogorov-Smirnov test.

The results of statistical tests for single constraint case, presented in Table 5.2, showed in all of the cases, all the methods have significant difference with each other based on the modified offline error (M_off_e) values. For the hyperplane translation, it is expected that as the magnitude of changes increases from small to large, the M_off_e values also increases as it gets harder for the algorithms to track the bigger changes. However, for multi-modal functions

TABLE 5.2: Statistical test results for single constraint setup

Function 1: Sphere							
Hyperplane translation	Frequency	Penalty		Feasibility		ϵ-Constrained	
		Rank	M_off_e	Rank	M_off_e	Rank	M_off_e
Small		2	123.47(±3.22)	1	51.09(±0.41)	3	82.02(±1.21)
Medium	1000	1	115.82(±2.78)	2	60.03(±0.48)	3	76.59(±1.38)
Large		2	185.07(±3.45)	1	158.70(±0.54)	3	131.53(±1.26)
Medium	2000	2	99.89(±1.86)	1	55.29(±0.21)	3	42.52(±0.55)
	500	1	130.548(±2.858)	2	66.45(±0.67)	3	110.88(±2.52)
Medium & rotation	1000	2	144.87(±2.92)	1	104.13(±0.57)	3	100.70(±1.40)
Function 2: Rastrigin							
Hyperplane translation	Frequency	Penalty		Feasibility		ϵ-Constrained	
		Rank	M_off_e	Rank	M_off_e	Rank	M_off_e
Small		1	342.56(±5.13)	2	74.51(±1.01)	3	276.78(±4.33)
Medium	1000	1	302.96(±4.83)	2	72.83(±0.88)	3	244.01(±3.53)
Large		2	281.67(±4.79)	1	155.14(±0.79)	3	221.16(±3.24)
Medium	2000	1	274.69(±3.72)	2	62.85(±0.41)	3	183.57(±3.38)
	500	1	327.74(±6.18)	2	88.30(±1.42)	3	295.96(±4.41)
Medium & rotation	1000	2	295.25(±4.07)	1	110.75(±0.82)	3	244.91(±4.20)
Function 3: Ackley							
Hyperplane translation	Frequency	Penalty		Feasibility		ϵ-Constrained	
		Rank	M_off_e	Rank	M_off_e	Rank	M_off_e
Small		2	5.59(±0.07)	1	4.65(±0.04)	3	4.53(±0.05)
Medium	1000	2	5.68(±0.07)	1	2.87(±0.04)	3	4.37(±0.06)
Large		2	4.99(±0.05)	1	2.35(±0.05)	3	3.69(±0.05)
Medium	2000	2	5.23(±0.08)	1	2.14(±0.01)	3	2.92(±0.04)
	500	2	6.12(±0.13)	1	4.03(±0.07)	3	5.56(±0.07)
Medium & rotation	1000	2	5.18(±0.05)	1	2.94(±0.04)	3	4.05(±0.04)
Function 4: Rosenbrock							
Hyperplane translation	Frequency	Penalty		Feasibility		ϵ-Constrained	
		Rank	M_off_e	Rank	M_off_e	Rank	M_off_e
Small		2	253460.47(±9060.06)	1	178489.77(±1984.36)	3	173310.42(±4686.65)
Medium	1000	1	247669.98(±7753.57)	2	182819.77(±1604.41)	3	198033.54(±2506.62)
Large		2	617497.34(±6471.45)	1	564491.21(±1645.71)	3	488869.17(±4354.94)
Medium	2000	1	230607.72(±4989.70)	2	184070.57(±963.10)	3	121674.39(±1650.03)
	500	1	265362.75(±7965.06)	2	180453.32(±1898.88)	3	243173.54(±5108.13)
Medium & rotation	1000	2	403920.26(±5611.91)	1	340830.23(±2661.17)	3	319687.74(±3933.93)

(Rastrigin and Ackley), this trend is not observable with the exception of feasibility for Rastrigin function. The reason behind this is that with small changes, algorithms are not able to come out of their previous local optima but with larger changes this will happen resulting to smaller M_off_*e* overall. For the experiment 2, all the algorithms for all the functions showed similar trend in which M_off_*e* increased as the frequency of changes increased. This is expected as with higher frequencies, algorithms have less time to reach to near optima solutions. Thus their overall deviation from the optima counted over all times increases (M_off_*e*).

Based on the results for M_off_*e*, the magnitude of the effect that hyperplane translation has on the performance of the algorithms is greater than the effect that frequency has. The magnitude of difference in M_off_*e* between the respective small and large settings was greater for hyperplane translation in every single test case. The reason behind this is that drastic changes lead to early solutions being infeasible or non-optimal, leading to larger M_off_*e*.

The third experiment tends to have one of the highest M_off_*e*, however, it is usually beaten by large hyperplane translation (large setting). This is because, in this experiment, the hyperplane is both translating and rotating (the *b*_value in this experiment is medium). The effect of hyperplane rotation on algorithm performance is lesser than that of translation, this leads to larger translation values affecting performance to a greater degree.

TABLE 5.3: Statistical test results for multiple constraint setup

Hyperplane translation	Frequency	Penalty		Feasibility		ϵ-Constrained	
		Rank	M_off_e	Rank	M_off_e	Rank	M_off_e
Function 1: Sphere							
Small		3	185.99(±7.13)	1	152.72(±0.68)	2	79.15(±4.38)
Medium	1000	3	346.88(±3.1763)	1	388.44(±0.88)	2	241.99(±2.63)
Large		3	474.22(±3.78)	1	563.38(±0.87)	2	361.96(±1.87)
Medium	2000	3	341.00(±3.81)	1	363.18(±0.40)	2	142.00(±0.93)
	500	3	347.31(±5.20)	1	419.49(±1.28)	2	307.56(±2.46)
Medium+ rotation	1000	3	437.82(±3.86)	1	511.31(±0.70)	2	337.00(±1.99)
Function 2: Rastrigin							
Hyperplane translation	Frequency	Penalty		Feasibility		ϵ-Constrained	
		Rank	M_off_e	Rank	M_off_e	Rank	M_off_e
Small		3	276.75(±9.08)	1	136.66(±0.63)	2	212.42(±2.46)
Medium	1000	3	231.74(±4.65)	1	351.49(±0.84996)	2	171.66(±2.41)
Large		3	201.14(±3.86)	1	504.28(±0.82)	2	140.25(±1.64)
Medium	2000	3	222.76(±4.93)	1	344.77(±0.39)	2	121.30(±2.10)
	500	3	237.96(±4.70)	1	371.18(±1.49)	2	210.80(±3.01)
Medium& rotation	1000	3	218.77(±3.48)	1	460.51(±1.09)	2	156.14(±2.15)
Function 3: Ackley							
Hyperplane translation	Frequency	Penalty		Feasibility		ϵ-Constrained	
		Rank	M_off_e	Rank	M_off_e	Rank	M_off_e
Small		3	2.93(±0.08)	1	3.32(±0.01)	2	2.19(±0.03)
Medium	1000	3	3.1001(±0.05)	1	1.92(±0.03)	2	2.01(±0.02)
Large		3	2.18(±0.02)	1	0.91(±0.01)	2	1.23(±0.01)
Medium	2000	3	2.90(±0.05)	1	1.55(±0.01)	2	1.37(±0.02)
	500	3	3.21(±0.04)	1	2.37(±0.04)	2	2.67(±0.03)
Medium & rotation	1000	3	2.49(±0.04)	1	1.33(±0.01)	2	1.53(±0.02)
Function 4: Rosenbrock							
Hyperplane translation	Frequency	Penalty		Feasibility		ϵ-Constrained	
		Rank	M_off_e	Rank	M_off_e	Rank	M_off_e
Small		3	671088.17(±30354.75)	1	455385.39(±3183.39)	2	334609.91(±15684.28)
Medium	1000	3	1422991.18(±8921.85)	1	1379550.37(±2485.46)	2	1050610.91(±11110.19)
Large		3	2098319.20(±8777.84)	1	2074607.43(±3342.87)	2	1647633.16(±7008.78)
Medium	2000	3	1411721.36(±14660.1552)	1	1280034.81(±979.24)	2	646148.54(±5147.41)
	500	3	1406504.59(±14776.65)	1	1494601.78(±3355.88)	2	1280330.18(±5866.88)
Medium& rotation	1000	3	1909408.88(±11346.38)	1	1895323.55(±3172.15)	2	1519570.30(±7386.32)

One of the limitations of M_off_*e* measure is that it is biased against generations where solutions are infeasible (considers worse solution of population in case of an infeasible solution). This makes our ranking procedure better suited to dynamic environments because it only uses an algorithm's best solution for each time and considers both criteria (objective value and sum of constraint violation) in selecting a higher performing result.

In the results, some algorithms are ranked higher than others despite having greater observed M_off_*e*. This discrepancy is caused by the ranking solutions selecting the single best solution for each time and then comparing the algorithms, whereas, M_off_*e* is measured per generation. Based on discrepancy of these two measures, ranking procedure will give higher priority to feasibility of the achieved solutions, while M_off_*e* is more in favour of the more closer to optima solutions. So an algorithm can be chosen based of which criteria is in our priority based of these two measures.

Based on ranking results, penalty is competitive with feasibility across the functions, despite usually having a higher M_off_*e*. Higher M_off_*e* values for penalty is because it accepts more infeasible solutions (based on our adaptive penalty method, this happens especially for the first generations) and M_off_*e* picks the worst solution if the best is infeasible. While, when ranking algorithms the best solution overall for each time is selected and this allows penalty to rank higher (as generations proceed penalty tend to increase

penalization factor to choose feasible solutions leading to final solutions more feasible). Achieving the third rank for ϵ-constrained also is expected based on its less strict behaviour toward infeasible solutions compared to feasibility. This becomes the reason to get more final infeasible solutions at each time, leading to its lower ranking compared to feasibility.

For Ackley function, regardless of the experiment, the algorithms ranked identically relative to each other. Feasibility outperformed the others, with penalty coming second and ϵ-constrained coming last. For the other functions, penalty and feasibility are struggling for the ranking. For the hyperplane translation and rotation, feasibility wins regardless of the tested function.

5.4.3 Multiple constraints

Statistical test for multiple constrained case, presented in Table 5.3, also shows all the methods have significant difference with each other. As the results in this table show, multiple constraint experiments tend to have higher M_off_e compared to their single constraint experiments counterparts. This is due to the increased difficulty that comes with satisfying multiple constraints at the same time over a single one. Regardless of the experiments and severities, in the rankings, feasibility is consistently as the best, the second ranking is for ϵ-constrained and the last one is for penalty. This is expected as in our ranking procedure the priority is with feasibility of the solutions. In the case of multiple constraints, in most of the experiments, algorithms are not able to find any feasible solution. As a result, based on the proposed ranking procedure, they will be ranked based on lower sum of constraint violations. Thus, feasibility based on its algorithm mechanism is usually the winner in this case.

5.5 Conclusion and future work

In this chapter, a framework has been proposed to generate benchmarks for testing algorithms in DCOPs. Our proposed framework can produce multiple benchmarks to be applied for testing any function and for any number of changes and dimension in the optimisation problem. The changes in the environment are imposed by translation and rotation of the hyperplane in single and multiple linear constraints. For testing our benchmark, three constraint handling techniques have been applied and compared (penalty, feasibility and ϵ-constrained) using DE algorithm. A procedure for ranking the algorithms, that is based on the feasibility rules, was proposed to analyse the results and compare the algorithms behaviour. Implementing different functions

showed that our proposed benchmark can be applied to test any function in DCOPs effectively. Moreover, the results showed that created changes had an observable effect on the performance of the compared algorithms. For future work, the proposed benchmark would need to be used with more advanced algorithms as the current constraint handling techniques struggled with dynamism.

Chapter 6

On the Use of Diversity Mechanisms in Dynamic Constrained Continuous Optimisation

6.1 Introduction

For avoiding premature convergence, that is a common problem in EAs, a diverse population is needed. Otherwise, there is no benefit of having a population; lack of diversity in population in the worst case, may lead the EA to behave like a local search algorithm, but with an additional overhead from maintaining many similar solutions [113]. Premature convergence in dynamic environments pose more serious challenges to EAs as when they are converged, they cannot adapt well to the changes. Indeed, having a diverse set of solutions in population helps to ensure the algorithm caters for changes in a dynamic environment. In the literature of EAs, diversity has been found to have various positive effects. To name a few, it is highly beneficial for enhancing the global exploration capabilities of EAs. It enables crossover to work effectively, improves performance and robustness in dynamic optimisation, and helps to search the whole Pareto front for evolutionary multiobjective optimisation [54]. In the related studies of runtime analysis [43, 113], diversity mechanisms proved to be highly effective for the considered problems, speeding up the optimisation time by constant factors, polynomial factors, or even exponential factors.

Diversity in EAs has been promoted through different approaches. A comprehensive classification is given in [29], that overall divides them into niching and non-niching approaches. Moreover, another classification is given based on the affected section of the algorithm: population-based, selection-based,

crossover/mutation-based, fitness-based, and replacement-based. These approaches have been applied in many different classes of optimisation problem so far including multi-objective [54], multi-modal [133], and constrained optimisation [28]. To name a few of them in DCOPs based on the aforementioned classification include: mutation-based [26], replacement-based [46] or population-based diversity mechanisms [18, 45].

However, regardless of the multiple benefits of diversity in EAs and in particular in dynamic optimisation, there is not an extensive study so far in DCOPs. What makes study of diversity in this problem class important is the contradictory effect of diversity and constraint handling techniques. While diversity mechanism spreads the solutions over the search space, the constraint handling technique has a tendency to guide the search toward feasible areas. The results of such study gives insight into the role of these opposing forces and their overall effect on algorithm's performance.

What we aim is to carry out a survey study over commonly used explicit diversity promotion methods (we exclude implicit methods that are via parameter tuning or selection mechanisms) investigating their effects in DCOPs. Our comparison aims to reveal which diversity promotion technique work better in each specific problem characteristic and why. Our investigations help to develop a better understanding of diversity role in DCOPs. The presented results reveal applying the diversity promotion techniques enhanced algorithm performance significantly based on statistical test applied in modified offline error values.

The remainder of this chapter is as follows. Experimental setup will be presented in Section 6.2. Results and discussion are reviewed in Section 6.3 and finally in Section 6.4 conclusions and future work are summarized.

6.2 Experimental setup

The classical benchmark, introduced in Chapter 2, is used to test the algorithms in this section. In the experiments, medium severity is chosen for the objective function ($k = 0.5$) and the constraints ($S = 20$). The other parameters are: frequency of change (f_c)=1000, runs=30, the number of considered times for dynamic perspective of the algorithm $5/k$ ($k = 0.5$) and the maximum number of evaluations $10^4 \cdot f_c \cdot 5/k$. Parameters of DE are chosen as population size $n_p = 20$, $CR = 0.2$, $F \sim \mathcal{U}(0.2, 0.8)$, and rand/1/bin is the chosen variant of DE.

6.3 Results and discussion

In this section, first the results for diversity measure is reviewed and then the results for MOF, $BEBC$, SR, and NFE values will be discussed.

6.3.1 Diversity results

Figure 6.1 illustrates the results for coefficient of variation of population (a measure for considering diversity explained in Chapter 2) for different methods per generation. Three functions have been opted for plots considering a range of different characteristics. Notice that the generations are not equal for all the methods. This is because the frequency of changes is mapped with the number of fitness evaluations, and some methods like Opp and RI use different number of fitness evaluations per generations compared to the other methods. Opp has almost half the number of generations when a change happens and RI is different from the other methods within a range (based on what the replacement rate is).

In general, RI shows almost the same trajectory regardless of the test case. It starts with maximum diversity around 0.6 and remains with a minor drop through the last generations. The lack of convergence in this method is due to random individuals inserted in the population at each iteration keeping diversity at a consistent level. For $No-div$ and CLS also an identical behaviour is observed regardless of the test case. They both start with a high value for diversity measure equal to 1 and converge to near zero after 45 generations which represents the number of generations in which the first change happens. Thus as diversity measure shows these two methods are not able to promote any diversity in population after they converge in the first change. As explained in Chapter 3, the way the tested algorithms react to changes is through re-evaluation of the solutions. For the other methods due to the applied diversity promotion technique, they can diverge faster leading to higher MOF values overall (is discussed in next section). But for these two methods finding new optimum after a change is very slow. This is because DE relies on differential vectors to maintain diversity, which are dependent on the population's diversity itself. So without increasing diversity extrinsically, diversity will remain low in them leading their inability to track new optimum.

For Opp, the diversity depends whether the opposite individual is accepted in the population or not. For test cases with smaller feasible regions, the diversity is low as the opposite individual is infeasible and hence it is rejected.

As if it is accepted, there is more diversity in the population, otherwise, this method behaves like No-div technique. For G24_7, the feasible area shrinks from 44.61% to 7.29% over time. This explains the behaviour of *Opp* in which at each change, the diversity increases sharply and then reduces gradually until next change happens. This pattern is repeated until around generation 160. From this generation afterwards, it loses its diversity as all of the opposite individuals fail to be chosen in the selection process (due to small feasible region).

Crowding is able to maintain high diversity over many generations in all test cases, thus keeping its effectiveness in responding to dynamic environment. This method shows higher diversity, near to 0.9, at the first generations and linger around o.4 until termination for functions G24_f and G24_7. For function G24_6b, its behaviour is slightly different as this test case has a special characteristic. For this test case, the objective function changes over time causing the global optimum to switch between two corners of the search space at each change step. This characteristic in this function can attribute the oscillated behaviour of diversity in *Fitnessdiv* method. By starting the first time before a change happen at generation 45, the diversity decreases. When the change happens, as the new optimum is in the other boundary of the search space, so the solutions must diverge again gradually to reach to the new optimum on the other corner of the search space until reaches to the next change in the environment and the same pattern repeats. For the other two test cases, *Fitnessdiv* showed a similar trend to *Crowding*, but with lower diversity. Another difference is after 150 generations, it looses its diversity. However, *Crowding* still keeps its diversity at around 0.4 until the end of generations. In general, as our selected measure of diversity is based on genotypic level, *Crowding* that is based of promoting diversity in genotypic level has got higher scores.

6.3.2 Statistical results

The results obtained for the compared algorithms using *MOF* values are summarized in Table 6.1. Furthermore, for the statistical validation, the 95%-confidence Kruskal-Wallis (KW) test and the Bonferroni post-hoc test, as suggested in [37] are presented (see Table 6.2). Non-parametric tests were adopted because the samples of runs did not fit to a normal distribution based on the Kolmogorov-Smirnov test.

Based on Tables 6.1 and 6.2, *MOF* shows significantly superior results for almost all of the methods compared to the base algorithm, $No - div$, for most

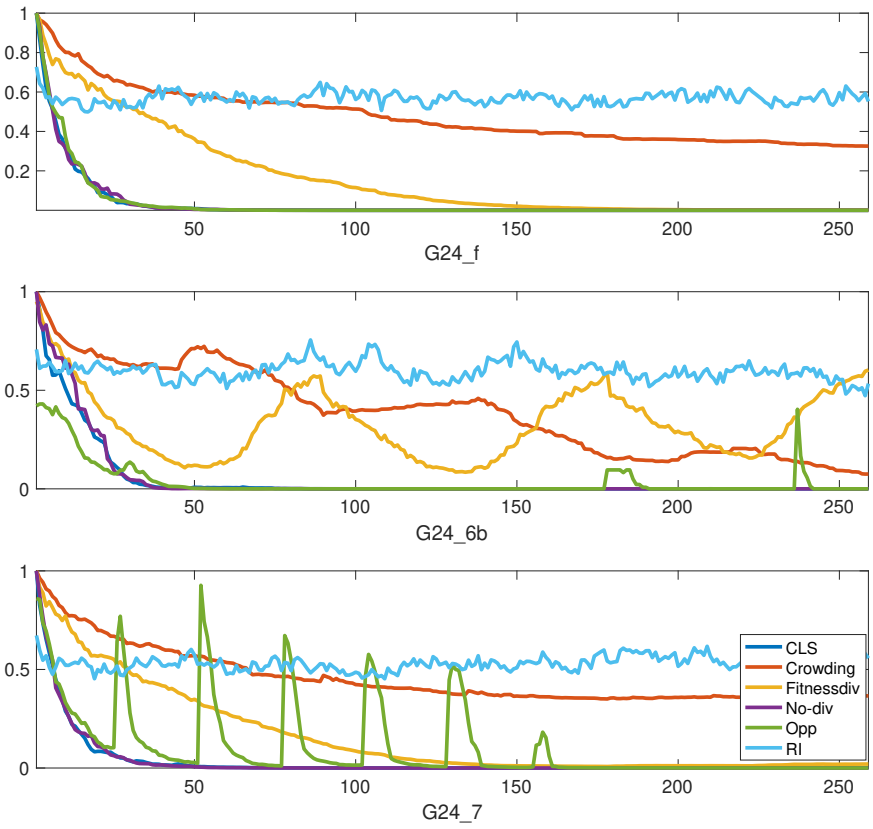


FIGURE 6.1: Y-axis: Diversity score (coefficient of variation of population), X-axis: Generations

of the test cases. Among them, *Crowding* has the highest frequencies of wins over the base algorithm compared to other methods. The difference of outperformance of *Crowding* compared to other methods is more significant in test cases by dynamic objective functions. This shows the other methods decrease in their performance dealing with dynamic objective functions compared to static ones, leading to bigger difference in dynamic cases.

On the contrary, the method that is very similar to $No-div$ method is CLS. As diversity results showed, this method is not increasing diversity that much, as it is more like a local search. This explains its superior performance compared to other methods in fixed test cases (G24_3f, G24_f). But it can not promote diversity that much (as only best solution at each iteration changes) and its inability to react to changes explains its higher *MOF* values.

RI preformed worse than the base algorithm for the problems with small feasible area (like G24_3f, G24_f, G24v_3, G24w_3) as there is a high probability that the inserted solutions are infeasible and hence they can not compete with the current best solution based on the applied constraint handling technique. On the contrary, for unconstrained ones and the ones with large feasible areas

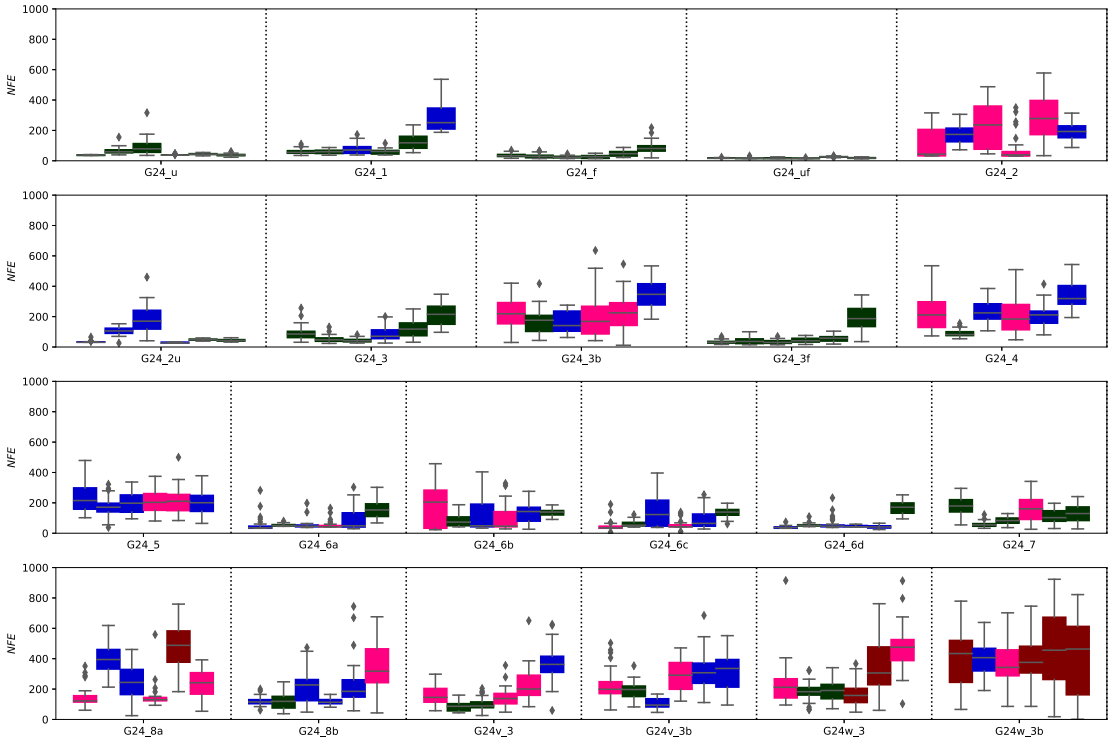


FIGURE 6.2: Boxplot of NFE values for $\epsilon = 10\%VTR$, color-coded with SR values: dark-red: $SR < 20\%$, purple: $SR \in [20\% - 50\%]$, blue: $SR \in [50\% - 80\%]$, dark-green: $SR > 80\%$. From left to right $1 = CLS$, $2 = Crowding$, $3 = Fitnessdiv$, $4 = No - div$, $5 = Opp$, and $6 = RI$.

showed the best results (like G24_u, G24_2, G24_2u, G24_6b, G24_8a). So generally, although the diversity of population is increased by inserting new solutions to the population, but because they are not feasible solutions they are not that effective. In addition, in this method the algorithm spends some more fitness evaluations (to the extent of replacement rate) compared to the base algorithm at each generation. Thus, the algorithm will have less computation budget for the evolution process itself as the changes happen after known number of fitness evaluations. In this benchmark there are some pairs of test cases that are used to test one behaviour of the algorithms like their abilities to handle constraints in the boundaries or the cases with disconnected feasible regions versus non-disconnected feasible area. Comparing the pairs of test cases with optima in constraint or search boundary over optima not in constraint or search boundary, with $No - div$ the optima in constraint boundary has got better results (G24_1, G24_2), (G24_4, G24_5), (G24_8b, G24_8a). While for other methods the trend is similar, for RI the trend is the other way, meaning with increasing diversity by inserting random solutions in RI, algorithm enhances its ability for finding optima which is not in constraint boundary. The pairs of fixed constraints versus dynamic constraints (G24_f, G24_7), (G24_3f, G24_3) show significant decrease in MOF values for dynamic constraints as it gets harder for the methods to deal with dynamism. The only exception

TABLE 6.1: Average and standard deviation of *MOF* values over 30 runs. Best results are remarked in boldface.

Algorithms	Functions					
	G24_u	**G24_1**	**G24_f**	**G24_uf**	**G24_2**	**G24_2u**
CLS	0.4331(±0.022)	0.5658(±0.014)	**0.0294(±0.012)**	0.0027(±0.002)	0.7965(±0.085)	0.3125(±0.058)
Crowding	0.0576(±0.027)	**0.0823(±0.025)**	0.0698(±0.029)	0.0033(±0.004)	0.2030(±0.049)	0.2488(±0.247)
Fitnessdiv	0.2084(±0.161)	0.5804(±0.014)	0.0394(±0.014)	0.0046(±0.004)	0.9292(±0.202)	0.8327(±0.293)
No-div	0.6215(±0.002)	0.5737(±0.014)	0.0332(±0.014)	0.0026(±0.002)	1.5727(±0.091)	1.5208(±0.002)
Opp	0.6118(±0.013)	0.5364(±0.080)	0.0446(±0.019)	**0.0020(±0.002)**	1.3765(±0.076)	0.0299(±0.015)
RI	**0.0344(±0.017)**	0.5183(±0.044)	0.4356(±0.040)	0.0031(±0.003)	**0.1935(±0.029)**	**0.0292(±0.011)**
	G24_3	**G24_3b**	**G24_3f**	**G24_4**	**G24_5**	**G24_6a**
CLS	0.2433(±0.058)	0.7861(±0.108)	**0.0263(±0.014)**	0.8449(±0.109)	0.7077(±0.099)	1.9470(±0.146)
Crowding	0.1618(±0.039)	**0.1556(±0.023)**	0.0471(±0.017)	**0.1204(±0.017)**	**0.1628(±0.033)**	**0.0734(±0.022)**
Fitnessdiv	**0.0745(±0.013)**	0.5126(±0.112)	0.0314(±0.009)	0.6216(±0.052)	0.8183(±0.134)	1.8504(±0.131)
No-div	0.3619(±0.152)	0.9496(±0.192)	0.0269(±0.011)	0.6756(±0.073)	1.2418(±0.094)	2.1163(±0.434)
Opp	0.3036(±0.117)	0.6724(±0.180)	0.0460(±0.020)	0.3271(±0.070)	0.9553(±0.041)	1.5525(±0.546)
RI	0.4339(±0.038)	0.5102(±0.037)	0.4079(±0.040)	0.5129(±0.043)	0.2170(±0.034)	0.2753(±0.075)
	G24_6b	**G24_6c**	**G24_6d**	**G24_7**	**G24_8a**	**G24_8b**
CLS	0.9702(±0.397)	1.7536(±0.458)	0.6860(±0.013)	0.3440(±0.094)	0.7243(±0.031)	0.6747(±0.028)
Crowding	0.2554(±0.064)	**0.0993(±0.050)**	**0.0775(±0.015)**	**0.1475(±0.026)**	0.5850(±0.030)	**0.1864(±0.039)**
Fitnessdiv	1.1505(±0.377)	0.9252(±0.523)	0.5438(±0.061)	0.1614(±0.032)	0.5200(±0.056)	0.7040(±0.069)
No-div	1.7479(±0.789)	1.9300(±0.646)	0.7887(±0.010)	0.3442(±0.162)	1.1064(±0.013)	0.7265(±0.007)
Opp	0.8291(±0.624)	0.8088(±0.696)	0.7756(±0.012)	0.2025(±0.148)	1.1914(±0.050)	0.7510(±0.032)
RI	**0.1876(±0.023)**	0.1882(±0.025)	0.2320(±0.039)	0.3904(±0.044)	**0.4755(±0.031)**	0.5557(±0.041)
	G24v_3	**G24v_3b**	**G24w_3**	**G24w_3b**		
CLS	0.5923(±0.507)	0.6941(±0.205)	1.0406(±0.503)	1.2456(±0.161)		
Crowding	**0.2529(±0.081)**	**0.2214(±0.057)**	0.5503(±0.198)	**0.6458(±0.141)**		
Fitnessdiv	0.2943(±0.147)	0.4219(±0.051)	**0.4784(±0.161)**	0.6905(±0.186)		
No-div	0.8776(±0.701)	0.9734(±0.222)	1.1860(±0.467)	1.2710(±0.215)		
Opp	0.7725(±0.719)	0.6778(±0.132)	1.1778(±0.371)	1.1999(±0.213)		
RI	1.4512(±0.101)	0.8169(±0.097)	1.2991(±0.091)	1.2263(±0.085)		

is the behaviour of *RI* method in which *MOF* values are almost the same in these two cases and this is attributed with stochastic nature of this method. The pairs of test cases for observing the effect of connected feasible region versus disconnected feasible region (G24_6b, G24_6a), (G24_6b, G24_6d) do not show any trend in the results. As this behaviour depends to the constraint handling technique of algorithms to a greater extent, that is similar in our case. *Opp* showed similar performance to the base algorithm in most of the cases with some exceptions. As the authors claim this method is to enhance convergence speed compared to base DE algorithm and is more effective in higher dimension problems. Our case is a two dimension, it is recommended to test this algorithm with higher dimension to see its effects in DCOPs.

The results of *BEBC* values also show *Crowding* has the best performance over the other methods in most of the test cases. In addition, for two test cases (G24_2, G24_8a) that *RI* showed better results in *MOF* values, based on *BEBC* values *Crowding* has better performance meaning *Crowding* end-up to closer values to optimum at the end of the change periods. For G24_uf, all the methods except *RI* reach to optimum values as the *BEBC* values are zero. Also for G24_u, *Crowding* and *RI* reach to optimum at all times.

Figure 6.2 shows the distribution of *NFE* values in box-plots. Boxes represent the values obtained in the central 25%-75%, while the line inside the box shows the median values. Whiskers depict highest and lowest values within

interquartile range and dots show outliers. The figure is color-coded based on SR values. The dark-red color shows SR values lower than 20%, the purple color belongs to $SR \in [20\% - 50\%]$, the blue color is for $SR \in [50\% - 80\%]$, and finally the dark-green belongs to SR values above 80%. In general for the test cases with fixed characteristics (G24_f, G24_uf, G24_3f) or unconstrained cases (G24_u, G24_2u) in most cases the algorithms manage to reach to VTR values in lower NFE and higher SR values. For the rest of the test cases, there is this general observation that shows regardless of the test case, *Crowding* and *Fitnessdiv* are more successful based on SR values that is easily observable based on stages that we defined and color-coded. Based on the colors, these two methods are usually one stage higher than the other methods in SR values.

Based on the characteristics of some functions, the distribution of NFE show quiet a high standard deviation in the results. These cases show less reliability in algorithms behaviour as in each run they achieve the VTR in different NFE values. Those belong to test cases with either smaller feasible areas or the test cases with specific characteristics (G24_2, G24_5). In these two test cases in some change periods the landscape is either a plateau or contains infinite number of optimum. In some cases the algorithms are almost unable to reach VTR in the given number of evaluations. The cases are colored with dark-red based of SR values. Figure 6.2 for RI shows in unconstrained cases it reaches to 10%VTR values with 100% of SR in almost of the cases but with high NFE values. In addition, for the functions with small feasible areas the results show a high number of standard deviation. Indeed, this was expected based on the random nature of this method. So depending on the random solutions inserted, in some cases, they can manage to track the optimum and in the others they are unable to do so. In general, RI is the most different method that has different trend compared to other methods in almost all the functions.

6.3.3 Discussions

Overall, results show *Crowding* that has higher diversity in population could reach to better MOF values. It also shows higher speed and frequency of reaching to 10%VTR values due to the results of SR and NFE values. Statistical test shows (Table 6.2) this method not only have significantly better results compared to base algorithm ($No - div$), but also compared to other methods in most of the test cases. Despite this method shows competitive results regardless of the test case, success of some methods highly depend on the tested problem. In this method, high diversity in population is created by avoiding genotypic similar individuals and without the need for extra

fitness evaluations. While for some other methods such as *RI* high diversity in population is achieved with the need for extra evaluations and the new solutions are inserted randomly. Although, the random insertion of solutions is often successful in unconstrained problems, in constrained problems it is dependant to the feasibility status of the new solutions and the applied constraint handling method. Indeed, the constraint handling technique forces the solutions toward feasible areas avoiding infeasible areas while diversity mechanisms tend to have solutions spread over the search space. Of course the severity of the competition depends to both opposing forces: diversity promotion technique and constraint handling technique applied. So depending to the applied methods, the created solution may not be accepted (rejected by constraint handling method) to be inserted to the population like in *Opp*, leading to low diversity in constrained problems; Or the solutions are inserted in the population increasing diversity (like in *RI*) but unable to compete with the best solution. Thus, one solution is applying adaptive constraint handling mechanisms. They can be designed in a way that at the first generations after a change, they increase their threshold (more relaxed with constraint violation) in order to allow diversity mechanisms explore the whole search space and then decrease to conduct the search toward feasible areas. The other suggestion is to use repair methods [3] as constraint handling technique, in which if the created solution by diversity handling mechanism is infeasible but a good solution (in terms of fitness function), preserve it in the population by repairing it. For more elaborate investigations on the roles of these two forces over each other and the overall algorithm performance, a measure that shows percentages of feasible and infeasible solutions that are selected for next generation can be helpful. As this measure can imply diversity handling mechanism ability to how it will balance exploiting feasible regions while exploring infeasible regions.

Since the results of *Fitnessdiv* are not as promising as *Crowding* for the problem types in this benchmark, this can infer that phynotypic diversity has less effectiveness in comparison to gynotypic diversity. For *CLS*, although an adaptive approach is used in such a way that at the first generations after a change, the local search length is large and gradually reduces; But as this local search is only applied to best solution at each generation, it is not enough to promote diversity as the results show. One solution is to do chaos local search randomly for some other individuals of the population besides to the best solution to increase diversity more. *RI* showed less reliability in *NFE* and *SR* values (based on high standard deviation in results) and its worse performance for *MOF* values for problems with small feasible area. Conversely,

TABLE 6.2: The 95%-confidence Kruskal-Wallis (KW) test and the Bonferroni post-hoc test on the *MOF* values in Table 6.1. The compared variants are denoted as: 1 = *CLS*, 2 = *Crowding*, 3 = *Fitnessdiv*, 4 = *No − div*, 5 = *Opp*, and 6 = *RI*.

Functions	Statistical Test
G24_u	1>2,4>1,1>6,4>2,5>2,4>3,5>3,3>6,4>6,5>6
G24_1	1>2,3>2,4>2,5>2,6>2,3>6,4>6,5>6
G24_f	2>1,6>1,2>3,2>4,6>2,6>3,6>4,6>5
G24_uf	
G24_2	1>2,4>1,5>1,1>6,3>2,4>2,5>2,4>3,3>6,4>6,5>6
G24_2u	4>1,1>5,1>6,3>2,4>2,2>5,2>6,3>5,3>6,4>5,4>6
G24_3	1>3,6>1,2>3,4>2,5>2,6>2,4>3,5>3,6>3,6>5
G24_3b	1>2,1>3,1>6,3>2,4>2,5>2,6>2,4>3,4>5,4>6,5>6
G24_3f	2>1,5>1,6>1,2>4,6>2,6>3,5>4,6>4,6>5
G24_4	1>2,1>3,1>5,1>6,3>2,4>2,6>2,3>5,4>5,4>6
G24_5	1>2,4>1,5>1,1>6,3>2,4>2,5>2,4>3,3>6,4>6,5>6
G24_6a	1>2,1>6,3>2,4>2,5>2,3>6,4>5,4>6,5>6
G24_6b	1>2,1>6,3>2,4>2,5>2,3>6,4>5,4>6,5>6
G24_6c	1>2,1>5,1>6,3>2,4>2,5>2,4>3,3>6,4>5,4>6,5>6
G24_6d	1>2,4>1,1>6,3>2,4>2,5>2,4>3,5>3,4>6,5>6
G24_7	1>2,1>3,1>5,4>2,6>2,4>3,6>3,4>5,6>5
G24_8a	1>3,5>1,1>6,4>2,5>2,2>6,4>3,5>3,4>6,5>6
G24_8b	1>2,4>1,5>1,3>2,4>2,5>2,5>3,3>6,4>6,5>6
G24v_3	1>2,6>1,4>2,5>2,6>2,4>3,6>3,6>5
G24v_3b	1>2,1>3,4>1,4>2,5>2,6>2,4>3,5>3,6>3,4>5
G24w_3	1>2,1>3,4>2,5>2,6>2,4>3,5>3,6>3
G24w_3b	1>2,1>3,4>2,5>2,6>2,4>3,5>3,6>3

this method is highly suggested to be used for unconstrained problems as it showed best results for these cases.

6.4 Conclusions and future works

Maintaining and promoting diversity in EAs is crucial to enable them adapt to dynamic environments in DCOPs. We have surveyed analysis of the diversity mechanisms, ranging from chaos local search, crowding, fitness diversity, opposition and random immigrants with a base DE algorithm for solving DCOPs. We have seen that diversity can be highly beneficial for enhancing the capabilities of DE for solving DCOPs. We found that diversity mechanisms that are effective for one problem may be ineffective for other problems, and vice versa. We observed that in some cases, the diversity mechanisms tend to have an opposing force towards the constraint handling technique. To gain more insights, comparing combination of diversity promoting mechanisms with different constraint handling techniques is worth to be studied. Another future study is to explore niching methods, such as clearing and fitness sharing, in the context of multi-modal optimisation. We did not include these methods in our study since our benchmark does not have any multi-modal problems.

Chapter 7

Neural Networks in Evolutionary Dynamic Constrained Continuous Optimisation: Computational Cost and Benefits

7.1 Introduction

The previous works on prediction mechanisms used as dynamic handling technique show how environmental change pattern can be extracted from the previous environments to provide effective guidance for the EA to predict the future optimum. For instance, in [106] the Kalman filter is adopted to model the movement of the optimum and predict the possible optimum in new environments. Similarly, in [110] linear regression is used to estimate the time of the next change and Markov chains is adopted to predict new optimum based on the previous times optimum. Likewise, in [136] the center points of Pareto sets in past environments are used as data to simulate the change pattern of the center points by using a regression model. In other works [58, 70], where the change pattern is not stable, it is proposed to directly construct a transfer model of the solutions/fitness, considering the correlation and difference between the two consecutive environments.

What is neglected in previous works is the time used for training and calling the predictor. In one recent work [69], the time spent for training the neural network (NN) is reported, however, it is not compared to the overall optimisation time. Such a comparison is needed, to reflect the overhead caused by using NN. In the relevant literature of dynamic problems, often a change is designed to happen after a constant number of fitness evaluations or generations [86]. But we need to consider the difference between the algorithm

using NN and the baseline algorithm in terms of the real computational cost. In some real-world problems [16], the condition that leads to the dynamic behaviour of the problem, happens after a time constraint (for instance prices are updated hourly in a power market). In this situation, we want an optimisation algorithm to achieve an optimum solution in a limited time budget, regardless of the number of fitness evaluations. In particular, time is important to be accounted when using a NN since by including several stages (data collection, training and predicting new solutions) can produce a noticeable time overhead in the optimisation. Therefore, we propose to create a change after an actual running time. With this, the time spent for training NN, is subtracted from the EA time. In consequence, all the methods have the same time budget for overall optimisation in each time. The purpose is to observe, considering the assigned time to NN that is indeed taken from the EA time for optimisation, if still NN helps the EA to improve the results.

Aside from the time constraint, our other concerns are regarding collecting sufficient samples to generalize predictions for new data, and the reliability of the samples. For those dynamic problems that the overall time horizon is short, we are not able to collect enough samples to train the NN in proper time. To alleviate this, we propose to consider more individuals on each time to speed up sample collection in shorter time steps. In problems with high frequency of changes, the solutions produced by EA at the end of each time are likely to be far from the real optimum. In such cases, using unreliable train data for the NN, in consequence, will produce unreliable predictions. Also, as the time spent for NN stays fixed regardless of the frequency, a higher frequency will mean a higher produced overhead by the NN in proportion to the EA.

Using differential evolution as our baseline, we experiment with different NN specifications. We explore how to introduce predicted solutions into population and the effect of the number of individuals introduced to be replaced on each change. The results of this chapter is published in European conference on artificial intelligence (ECAI 2020) [48]. The remainder of this chapter is as follows. Experimental setup will be presented in Section 7.2. Experimental results are reviewed in Section 7.3 and finally in Section 7.4 the results are concluded.

7.2 Experimental setup

In this section, designed experiments, the test problems and the applied parameters are reviewed.

7.2.1 Designed experiments

Regarding to integration of NN with DE algorithm, there are a couple of experiments designed as follows.

- **Frequency changes:** In this experiment, we observe how the frequency of changes will affect the results. The frequency of change, denoted by τ, represents the width that each time lasts. Notice that when we refer to higher frequencies of change, we mean lower values for τ, since higher frequencies of change happens when there is shorter time interval between each change (τ). Three frequencies of change: 0.5, 1 and 4 will be used to experiment with high, medium and low environmental changes respectively. As mentioned, in this work the real time is considered, so the values above represents time in seconds between the two consecutive changes. To have an idea, these values represent the following number of fitness evaluations: $0.5 \approx 1000$, $1 \approx 2000$, $4 \approx 9000$. Undoubtedly, these numbers are not constant for all test cases due to different time-complexity of each function and stochastic nature of EA.
- **Building train data set:** In this experiment, we explore the effect of using more individuals (k-best) of population at each time for training the NN. We change the parameters of NN like batch size, epochs, and number of samples accordingly to have roughly the same timing budget for NN with respect to the overall time in each case. In the case for one individual ($k = 1$), we do not limit the overall sample size, so as time increases, the samples aggregate. In other words at every time, NN is trained with all previous times best individuals. The reason is as we consider one individual at each time, the collected samples are a few; hence in order to have a reasonable number of samples we keep the previous samples. Conversely, for $k > 1$ case, we use a window as the samples aggregation limit window (denoted as n_w=5). For this case, we limit the number of samples since otherwise as the time increases, they will exponentially increase. In such case, as we have a constant budget then the time assigned to the EA decreases severely.
- **Number and mechanism to insert predictions:** In this experiment, number of individuals to be replaced (denoted by n_p) with predicted solutions are varied and tested. More number of predicted individuals are created by adding noise to the one predicted value by NN. In this experiment, the added noise is constantly at 10% of the variable boundaries. However, experimenting the noise effect on the results can be considered in a future study. In addition to the number of replaced

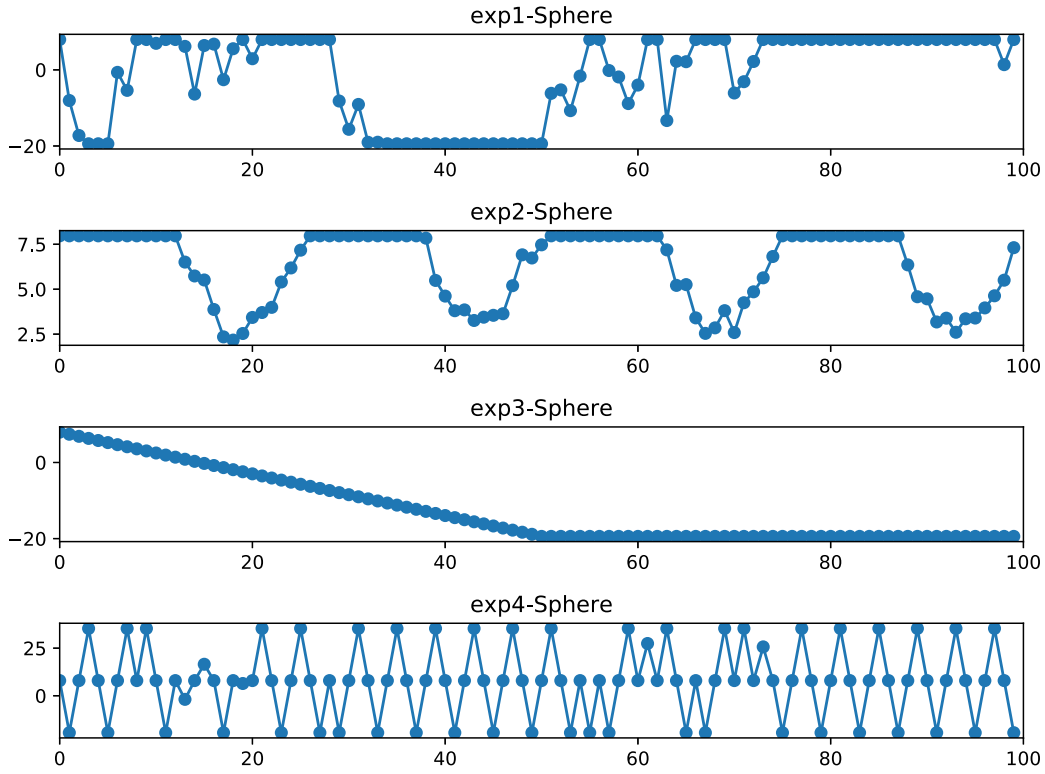


FIGURE 7.1: PCA plot of best_known positions for each experiment over time

individuals, two different replacement approaches are also compared. The first one, denoted by NNR, replaces randomly chosen individuals of population with the predicted solutions. The second one, denoted by NNW, first ranks the individuals of population and then replaces the top worst among them with the predicted solutions.

7.2.2 Test problems and parameters settings

We applied the test problem introduced in chapter 2 for neural network experiment. Figure 7.1 shows the pattern in which the position of optimum changes in each experiment[1], using principal component analysis (PCA) method to map the thirty dimension to one dimension scale. Feasibility rules [35] is applied for the constraint handling. For change-detection approach we used re-evaluation of the solutions. As change reaction mechanism, two approaches are considered. In the first approach, called noNN, the whole population is re-evaluated. In the second approach, some individuals of the population will be replaced with the predicted solutions (randomly selected or worst solutions of the population) and the rest of the individuals are re-evaluated.

[1]The results belong to best_known solutions of each time retrieved by executing 100,000 runs of our baseline DE algorithm.

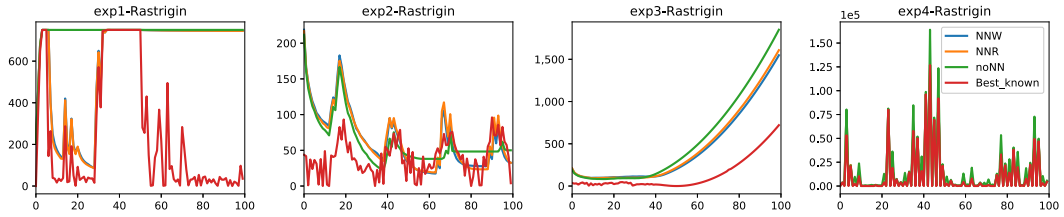


FIGURE 7.2: Fitness values of Rastrigin for $\tau = 1$, color-coded with each method over time

The other parameters are: frequencies of change τ= 0.5, 1, 4; problem dimension=30, runs=30 and the number of changes or times=100. Parameters of DE are chosen as $NP = 20$, $CR = 0.2$, $F \sim \mathcal{U}(0.2, 0.8)$, and rand/1/bin is the chosen variant of DE [7]. Variable boundaries are limited in $x_i \in [-5, 5]$. Parameters of NN are different based on number of individuals considered to build training data: case $k = 1$: epochs=10, $n_w = \infty$ (e.g. all previous times are considered) and case $k > 1$: epochs=3, $n_w = 5$. Also in both cases, we use batch_size=4, min_batch=20 and $n_p = 3$. All the experiments were run on a cluster, allocating 1 core (2.4GHz) and 4GB of RAM. Our code is publicly available on GitHub: $https://github.com/renato145/DENN$.

7.3 Experimental results

In this section, the main findings about designed experiments explained in Section 7.2 are presented.

7.3.1 Frequency changes

For most experiments and functions, by increasing τ, MOF values decrease, presented in Figure 7.3. However, there are some exceptions: for all functions using noNN (exp1 and exp4) and also for Rastrigin function in all methods (exp3). Looking to PCA plot of the optimum positions in Figure 7.1 for exp4 (and exp1 for some changes), the optimum alters drastically between two consecutive changes. As noNN only reevaluates the solutions when a change is detected, they are far away from new optimum and lacking a diversity promotion technique to aid exploring other regions of the search space lead to higher MOF values. However, this is only happening in $\tau = 4$ as the solutions are more converged in this case compared to the other τ values. As for exp1, drastic changes repeat less often, the drop in performance of MOF value is less severe compared to exp3 (drop in values as τ increases). The best fitness values achieved by each method are presented in Figure 7.2 over time. From this figure, it is also observable that in most functions and

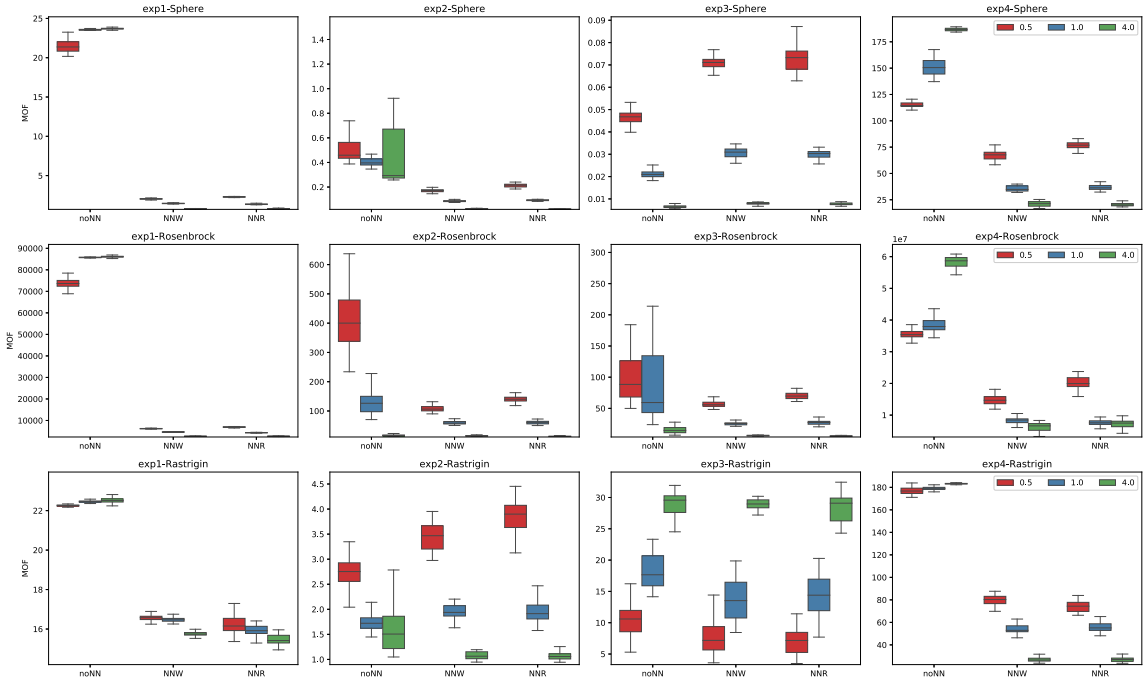


FIGURE 7.3: Distribution of MOF values for each method color-coded with τ for 30 runs

experiments, the best value achieved by NN variants is tracking the optimum more closely. Figure 7.4 illustrates the overall performance of the methods compared to each other color-coded with different functions considering their performance on all the frequencies. Overall comparison of methods is not easily possible with MOF values as they are not of the same scale. So we use another measure denoted as MOF_norm that enables an overall comparison of methods as they represent standard MOF values (Figure 7.4). To achieve standard values in each set of function and experiment, the values are divided by the minimum value among all methods. So the method with lowest MOF value has MOF_norm value equal to one and the others are proportionally calculated.

From aforementioned figures (7.2, 7.3, and 7.4) it can be observed that NN variants show their best performance for the experiments where there is a trend in the position changes. Looking to PCA plots (see Figure 7.1) for exp3 until the time around 50, we have a linearly decreasing trend and from then it is saturated in variable boundary remaining constant. As the training data for NN depends to previous behaviour of the algorithm, it is unable to self-improve as time passes. The NN variants can obtain better results even in exp1 without a consistent trend. In this experiment, as we consider 5 previous times to train the NN ($n_t = 5$), for this n_t there is not a consistent trend observable. The better results achieved is partly because the newly generated solutions can increase diversity (as our baseline algorithm lacks a

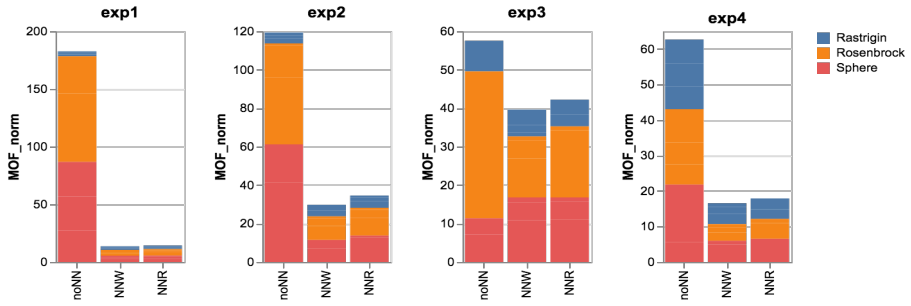
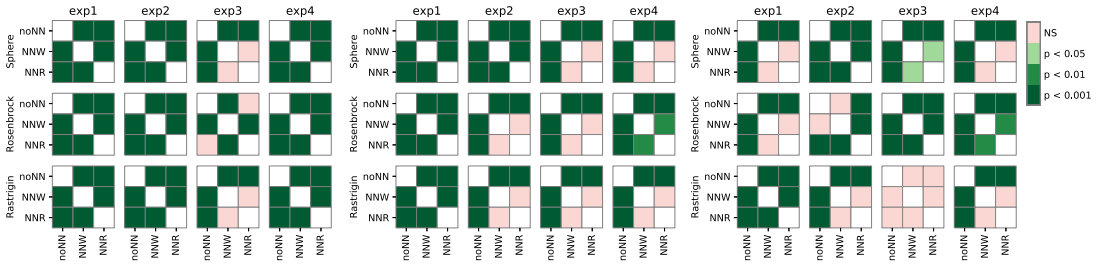


FIGURE 7.4: MOF-norm values considering all frequencies



(A) $\tau = 0.5$ (B) $\tau = 1$ (C) $\tau = 4$

FIGURE 7.5: Kruskal-Wallis statistical test on MOF values for different frequencies

proper diversity mechanism to be activated when a change happens). Thus, even though the change pattern is not fully consistent, but for the algorithm without other proper mechanism for reacting to changes still can improve the results. In addition, this is the reason the difference between MOF values (Figure 7.3) for this experiment between noNN and NN variants is more significant. Figure 7.2 also shows for this frequency, the optimum is not tracked closely for noNN. However, for τ=0.5, as still population has a fair amount of diversity, the optimum is tracked more closely.

To validate the results, the 95%-confidence Kruskal-Wallis statistical test and the Bonferroni post hoc test, as suggested in [37] are presented. Nonparametric tests were adopted because the samples of runs did not fit to a normal distribution based on the Kolmogorov-Smirnov test. Figure 7.5 shows a heat-map of the test results on MOF values. In this figure, as the legend represents, the pink squares show the methods with not-significantly different (NS) results, and the squares in the spectrum of the green colors show the significantly different methods with the mentioned p-values. Results show in most test cases in different frequencies, the methods have significant difference to each other. However, for higher τ values (1 and 4) the NN variants show similar behaviour for almost half of the test cases. The reason is as the solutions are converged in high frequencies, there is not significant difference between replacing the worst solutions or select them randomly.

Figures 7.6 and 7.7 show a boxplot of the ARR and SR values respectively,

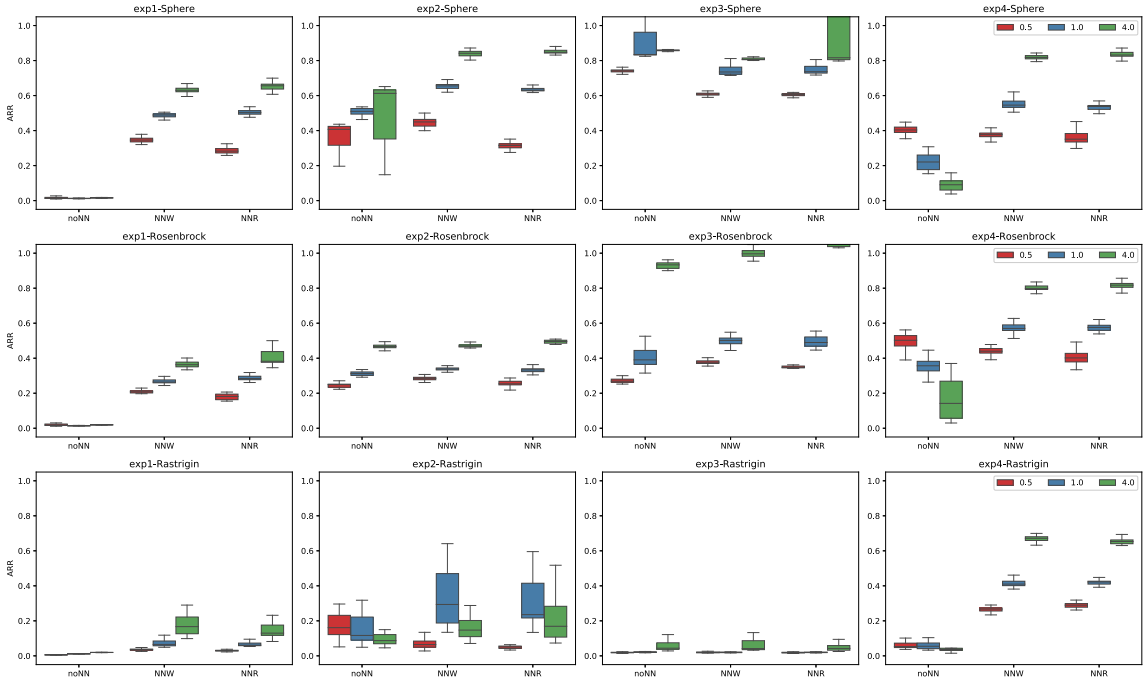


FIGURE 7.6: Distribution of absolute recovery rate (ARR) values color-coded with τ for 30 runs

for different methods and frequencies of change. NN variants in most experiments and functions show better ARR values; meaning they can recover faster after a change. In addition, SR values show better results for NN variants; meaning they can reach to an ϵ-precision (=10%) of optima for more changes (or times) compared to the method without using prediction. When comparing each method for different frequencies, there is this general trend that better results are achieved as we proceed from frequency 0.5 to 4, as the algorithms have more timing budget to get better results. In addition, NN variants in this frequency, are trained with more precise data as EA has more timing to achieve better solutions.

Table 7.1 represents the percentages of the amount of time spend for calling NN unit compared to overall optimisation time. Regardless of the experiment and function, the results for $\tau = 0.5$ show around 20-25%, $\tau = 1$ around 10-12% and $\tau = 4$ around 3%. This shows when τ is higher, it is more cheap to use NN in terms of the computational cost. When τ is low the proportion of the time for doing optimisation itself is lower, hence, the samples used to train NN do not represent real optimum or near optimum values and the prediction from NN is not exact in consequence. For this, in most test cases the difference of the performance of NN variants in $\tau = 0.5$ and $\tau = 4$ is bigger compared to noNN method.

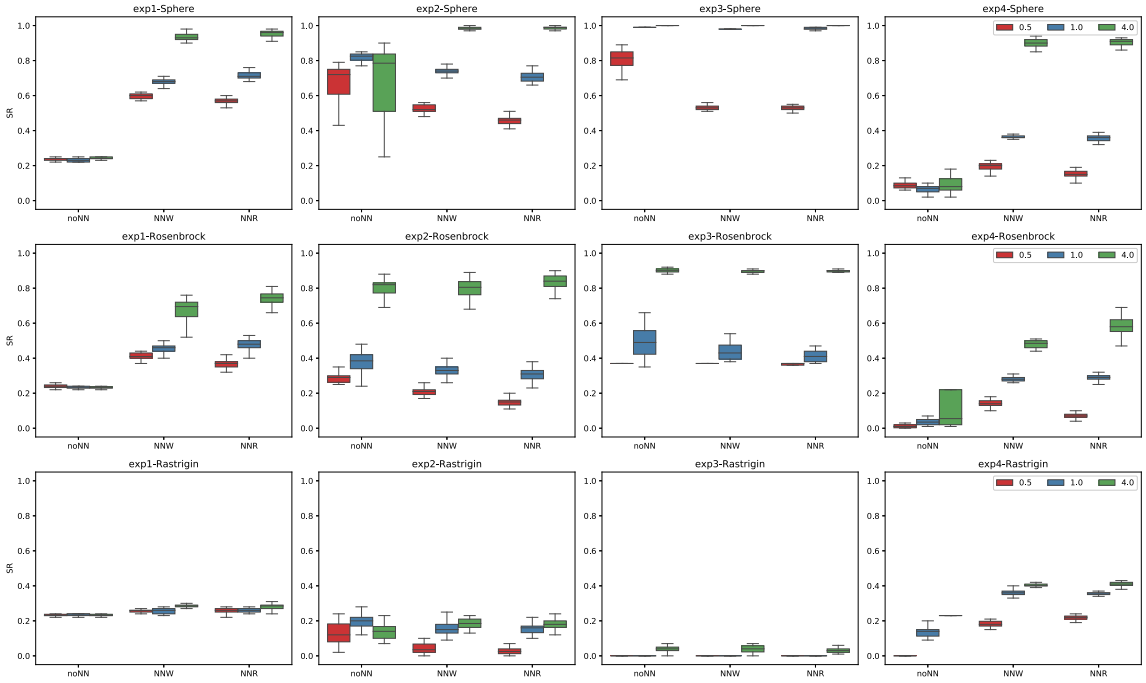


FIGURE 7.7: Distribution of success rate (SR) for 30 runs; number of times algorithms reach to 10% of the vicinity of optima values per overal times

TABLE 7.1: NN-time; time spend for training and using NN in proportion to overall optimisation time (mean ± std: for 30 runs)

experiment		exp1			exp2			exp3			exp4		
freq		0.5	1.0	4.0	0.5	1.0	4.0	0.5	1.0	4.0	0.5	1.0	4.0
Sphere	NNW	0.24 (±0.00)	0.11 (±0.00)	0.02 (±0.00)	0.25 (±0.00)	0.10 (±0.00)	0.03 (±0.00)	0.24 (±0.00)	0.12 (±0.00)	0.03 (±0.00)	0.19 (±0.00)	0.10 (±0.00)	0.03 (±0.00)
	NNR	0.21 (±0.02)	0.12 (±0.03)	0.03 (±0.00)	0.22 (±0.03)	0.12 (±0.01)	0.03 (±0.00)	0.24 (±0.01)	0.11 (±0.00)	0.03 (±0.00)	0.19 (±0.00)	0.12 (±0.00)	0.02 (±0.00)
Rosenbrock	NNW	0.23 (±0.00)	0.12 (±0.00)	0.02 (±0.00)	0.22 (±0.00)	0.11 (±0.00)	0.02 (±0.00)	0.23 (±0.00)	0.11 (±0.00)	0.03 (±0.00)	0.22 (±0.00)	0.11 (±0.00)	0.02 (±0.00)
	NNR	0.20 (±0.02)	0.12 (±0.02)	0.03 (±0.00)	0.21 (±0.02)	0.12 (±0.01)	0.03 (±0.00)	0.24 (±0.01)	0.11 (±0.00)	0.03 (±0.00)	0.19 (±0.01)	0.12 (±0.00)	0.03 (±0.00)
Rastrigin	NNW	0.20 (±0.00)	0.14 (±0.00)	0.03 (±0.00)	0.20 (±0.00)	0.12 (±0.00)	0.03 (±0.00)	0.24 (±0.00)	0.13 (±0.01)	0.03 (±0.00)	0.19 (±0.00)	0.11 (±0.00)	0.03 (±0.00)
	NNR	0.19 (±0.04)	0.12 (±0.02)	0.03 (±0.00)	0.20 (±0.02)	0.12 (±0.01)	0.03 (±0.00)	0.24 (±0.00)	0.13 (±0.01)	0.03 (±0.00)	0.19 (±0.01)	0.11 (±0.00)	0.03 (±0.00)
Mean values		0.21	0.12	0.03	0.22	0.12	0.03	0.24	0.12	0.03	0.2	0.11	0.03

7.3.2 Building train data set

We tested 1, 3, 7 and 9 individuals (k-best) to be used to train NN. As Figure 7.8 represents, one individual ($k = 1$) has not showed good performance based on MOF_norm values. The reason is the slow sample collection leads to non-promising MOF values. Due to our min_batch size (=20), our first prediction is possible at change (time) 26. On the other hand, the results for 9 individuals also degrade. For building our sample data, we take a random combination of solutions for k-best solution of each time. Therefore, if the diversity of population is high, the first best solutions are distant from one another and consequently might not represent the change pattern of the optimum correctly.

Overall, too few or high number of individuals is not a proper choice. So for the rest of the experiments, $k = 3$ is chosen to feed NN trainer. Due to space limitation, for this and next experiment, we exclude exp3 to base our conclusions on the experiments where NN performed more promising.

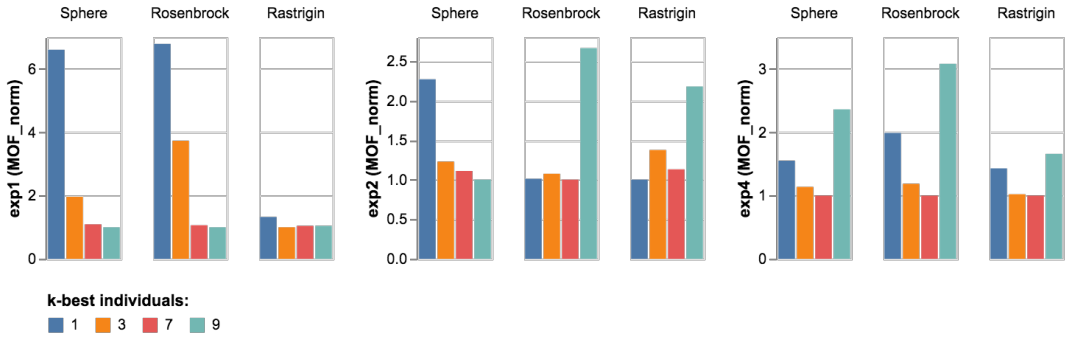


FIGURE 7.8: k-best individual selection for building NN samples

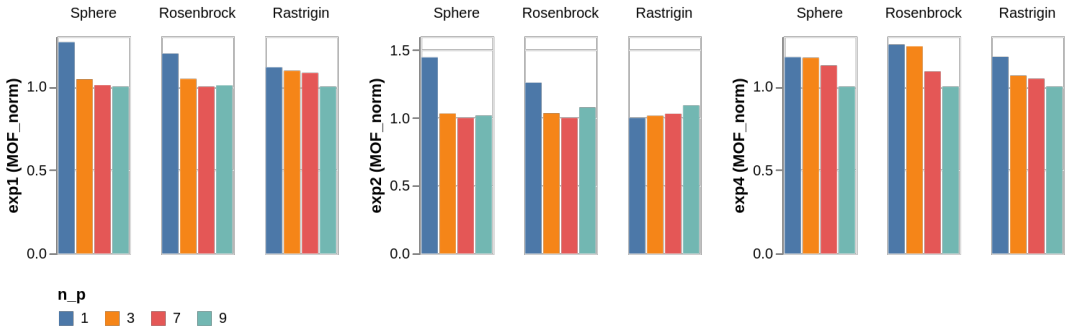


FIGURE 7.9: Number of individuals of population replaced by predicted solutions

7.3.3 Number and mechanism to insert predictions

When we use a small n_p, the effect that NN have in the overall optimisation is minor. On the other hand, using a high n_p will decrease the diversity of the population, given that all the individuals to be included are centered around the same predicted solution with a small added noise (10% of the variable boundary). We expect this decrease of diversity to adversely affect the results. However, on our experiments using high n_p values, we do not always observe such a behaviour, as can be seen in exp1 and exp4 (see Figure 7.9). Also, looking to the pattern of the changes for exp4 (see Figure 7.1), the position changes drastically between two alternative times. Since for the rest of the population we only reevaluate the solutions, thus replacing more individuals will help to transfer the population to a new region of the search space. This is because our baseline algorithm does not promote any diversity, as it only reevaluates the solutions when a change happens. Hence, replacing more individuals, particularly for the case with correct predictions, does not have an adverse effect. However, we believe that in cases where extra mechanisms to promote diversity are considered, the decrease of diversity generated by choosing a high n_p will decrease the overall performance. In general, there is not significant difference in the results of MOF values when using $n_p > 1$.

Regarding to replace mechanism, based on the results for MOF values shown

in Figure 7.4, we can observe in general that NNW shows better performance than NNR. The difference is clearly seen for τ= 0.5, as seen in Figure 7.5a, where there is a significant difference between these two methods for 10 out of 12 test cases. For larger values of τ (1 and 4), on the other hand, approximately half of the test cases show significant difference. The reason for this is the small distance between worst and random picked solutions. As with a higher τ, all individuals in population are likely to have converged close to the same optimum position. To conclude, we suggest to insert the predicted solutions by replacing the worst solutions of the population.

7.4 Conclusions and future works

We studied the behaviour of using a NN together with DE for solving DCOPs. Considering generated overhead by NN, we observed when the frequency of changes is high, the time spent for NN becomes more noticeable in proportion to overall time. In addition, due to shorter time between changes, the optimisation algorithm might not achieve good solutions. In this case, the collected data is not helpful for the prediction or even becomes misleading for the optimisation algorithm. In our experiments, for high frequencies of change, NN variants showed their worst results.

Moreover, for the algorithms integrating with NN enough training data is needed. Hence, for short overall time horizons, this might not be an efficient method as for the first change periods, we need to collect data. Moreover, training a NN with small amounts of data will overfit the NN, making it difficult to generalize and make predictions for new data. The proposed method to collect more individuals of population from each time to train NN, will lead to make NN ready faster but this is possible when there is lower diversity in population. If the population is diverse, the first best solutions will have higher distance and might not be a good data to train the network. In general, we observed that diversity has a significant role when applying prediction methods in DCOPs. For replacing predicted solutions, we observed when we have diversity among solutions, selection of n_p worst solutions performed better than selecting them randomly. In general we believe controlling diversity besides prediction methods is essential. To do so, and for a better understanding of the behaviour of the prediction it is suggested to check prediction error and based on that, diversity mechanisms be applied properly together with prediction. We observed in some experiments the lack of diversity lead to poor results, while a basic diversity mechanism could improve results, particularly when predictions are wrong. One suggestion for future work is to define an

adaptive parameter that considers the prediction error to control to which extent to use diversity mechanisms. The other future work is to explore the effect of the noise added to the predicted solutions on the final performance of the methods. Less noise indicates relying more on the results of the predicted solution. Perhaps we can have an adaptive noise, that varies based on the results of the prediction error.

Chapter 8

Neural Networks and Diversifying Differential Evolution

8.1 Introduction

Among the many approaches proposed for reacting to the changes in dynamic environments, diversity mechanisms [21, 45] are the simplest and most popular. In Chapter 6, we observed how common diversity mechanisms can significantly enhance the performance of a baseline differential evolution (DE) for different environmental changes [49]. Other approaches include memory-based approaches [99], multi-population approaches [17] and prediction methods [18]. Previous work on prediction approaches has shown that they can be well suited to dealing with dynamic problems where there is a trend in the environmental changes [72]. For instance, in [106], a Kalman filter is applied to model the movement of the optimum and predict the possible optimum in future environments. Similarly, in [110], linear regression is used to estimate the time of the next change and Markov chains are adopted to predict a new optimum based on the previous time's optimum. Likewise, in [136], the centre points of Pareto sets in past environments are used as data to simulate the change pattern of those centre points, using a regression model. Besides these methods, neural networks (NNs) have gained increasing attention in recent years [58, 70, 72, 73]. In [73], a temporal convolutional network with Monte Carlo dropout is used to predict the next optimum position. The authors propose to control the influence of the prediction via estimation of the prediction uncertainty. In [72], a recurrent NN is proposed that is best suited for objective functions where the optimum movement follows a recurrent pattern. In other works [58, 70], where the change pattern is not stable, the authors propose directly constructing a transfer model of the solutions and

fitness using NNs, considering the correlation and difference between the two consecutive environments.

However, despite previous attempts, there are still some concerns regarding the application of NNs to the evolution process. As integrating NNs in EAs is more complicated than using standard diversity mechanisms, the question arises as to whether they enhance the results to an extent that compensates for their complexity. In addition, previous work has mainly compared prediction-based methods with a baseline algorithm and other prediction-based methods [48, 72]. To the best of our knowledge, only one recent work considers other mechanisms for dynamic handling in comparison with prediction [73].

However, the time spent by NN has not been accounted for. We believe that to compare NN with other standard methods fairly, the relative time consumption of different methods needs to be accounted for, as this may create a noticeable overhead in the optimisation process; time costs can occur across the following stages: data collection, training and prediction of new solutions. To account for the timing used by NN, we create a change after an actual running time of the algorithm. This is not the usual technique; in the dynamic problem literature, a change is often designed to happen after a number of fitness evaluations or generations [86]. But such a method fails to account for the time spent by NN. Therefore, to evaluate the effectiveness of NN in the described setting, this chapter compares common diversity mechanisms using a DE algorithm with and without NN. In this study, we try to answer the following questions, taking into account the time spent on NN:

- How is NN comparison with other simpler mechanisms for diversifying DE in order to handle dynamic environments?
- Does diversity of population in DE play a role in the effectiveness of NN?
- Do different frequencies of change impact the suitability of NN in comparison to diversity mechanisms?

The results of our study show that the extent of the improvement in the results of tracking the optimum (speed and error of tracking), when integrating the neural network and diversity mechanisms, depends on the type and the frequency of environmental changes. In addition, we observe that having a sound diversity in the population has a significant impact on the effectiveness of NN. The remainder of this chapter is as follows. Our experimental methodology is presented in Section 8.2. In Section 8.3, a comparison across all the

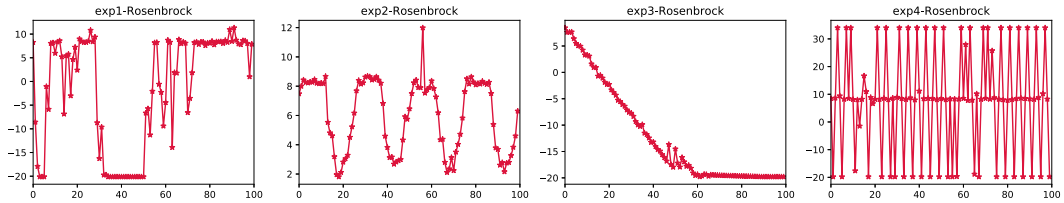


FIGURE 8.1: PCA plot of best_known positions for each experiment over time

methods is presented. In Section 8.4, we carry out detailed experimental investigations on the use of NNs in different variants of diversifying DE. Finally, in Section 8.5, we finish with some conclusions and elaborate directions for future work.

8.2 Experimental methodology

In this work, two approaches are considered as change reaction mechanisms with DE. In addition to these change reaction mechanisms diversity mechanisms are considered. In the first approach (noNN), the whole population is re-evaluated. In the second approach (NN), a number of the worst individuals (in terms of objective function) in the population are replaced with the predicted solutions, and the rest of the individuals are re-evaluated. We applied the most common diversity mechanisms, explained in Chapter 3: crowding (CwN), random immigrants (RI), restart population (Rst) and hyper-mutation (HMu). For a review regarding the effect of diversity mechanisms in dynamic constrained optimisation, see Chapter 6 [49].

The test problem for neural networks explained in Chapter 5 is applied. Figure 8.1 shows the pattern in which the position of optimum changes for Rosenbrock function in each experiment [1]. To achieve this plot, the principal component analysis (PCA) method is used to map the thirty dimensions to a one-dimensional scale. The frequency of change, denoted by τ, represents the width for which each time lasts. We indicate that when referring to higher frequencies of change, we are talking about lower values of τ, since higher frequencies of change happen when there is a shorter time interval between each change (τ). Different frequencies of change (1, 5, 10 and 20) will be tested. As mentioned earlier, this work will consider wall clock time. As such, the values above represent time in seconds between the two consecutive changes. To provide an idea based on number of fitness evaluations, these values represent the following number of fitness evaluations for the baseline algorithm:

[1] The results belong to best_known solutions of each time, retrieved by executing 100,000 runs of a baseline DE algorithm.

$1 \approx 2000$, $5 \approx 11000$, $10 \approx 22000$, $20 \approx 45000$. Undoubtedly, these numbers are not constant for all test cases, due to the differing time-complexity of each function and the stochastic nature of DE. The other parameters are: $d = 30$,

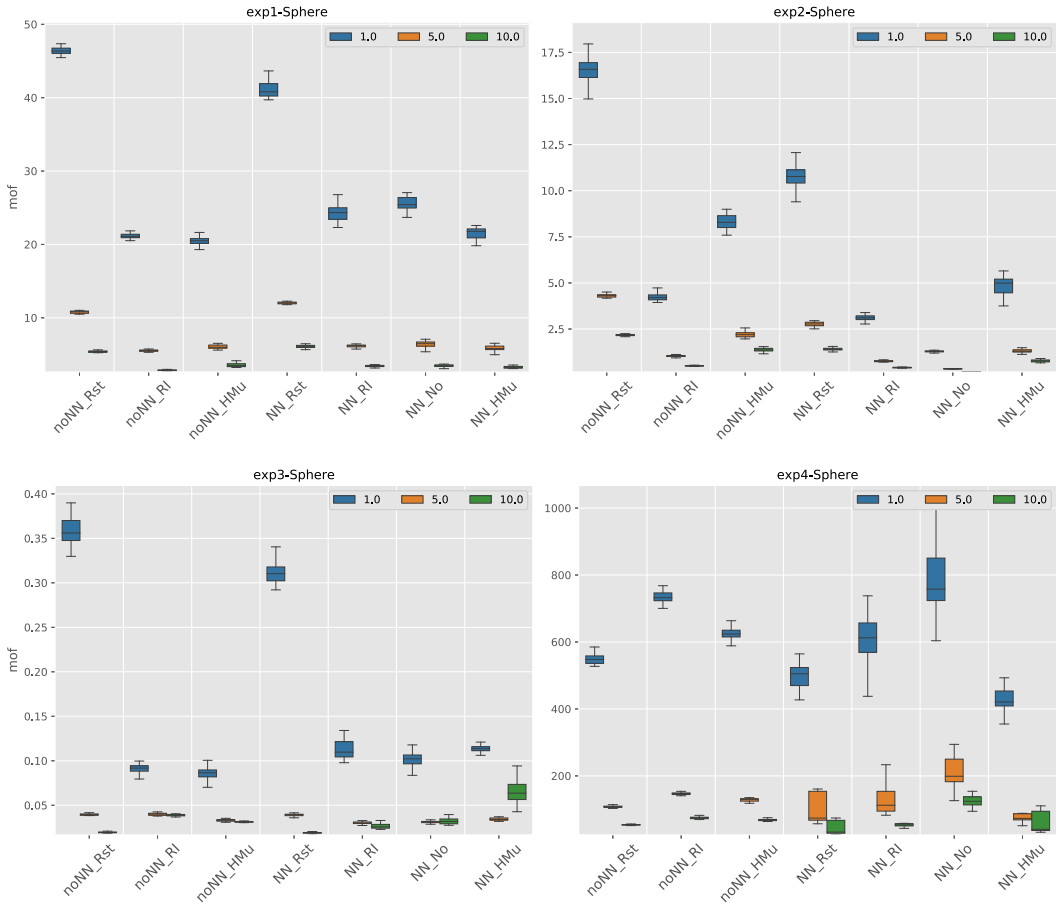


FIGURE 8.2: Distribution of MOF values for 20 runs, different frequencies (τ). Lower values represent better performances

runs = 20 and the number of changes or times = 100. Parameters of DE are as follows: $NP = 20$, $CR = 0.3$, $F \sim \mathcal{U}(0.2, 0.8)$ and rand/1/bin is the chosen variant of DE [7]. The solution space is within $[-5, 5]^d$. For the RI and HMu methods, the replacement rate for noNN methods is 7, and for NN methods, it is 2. Since we insert five individuals with NN, we want to insert a constant number of individuals in each case overall. For the HMu method, F and CR are changed for a number of generations (depending on τ) to $F \sim \mathcal{U}(0.6, 0.8)$ and $CR = 0.7$, then, after this generation (6τ), they return to normal.

Parameters of NN were selected in a set of preliminary experiments [48]: $k = 3$: epochs = 4, $n_w = 5$, batch_size = 4, min_batch = 20 and $n_p = 5$. All the experiments were run on a cluster, allocating one core (2.4GHz) and 4GB of RAM. Our code is publicly available on GitHub: $https://github.com/renato145/DENN$.

TABLE 8.1: Pairwise comparison of methods on MOF values for $\tau = 1$ and 20 (mean of 20 *runs*)

Experiment	Function	noNN_RI	NN_RI	noNN_HMu	NN_HMu	noNN_No	NN_No	noNN_CwN	NN_CwN	noNN_Rst	NN_Rst
$\tau = 1$											
1	Rastrigin	**100.59**	103.36	**105.7**	113.28	517.9	**489.26**	**222.97**	376.18	144.39	**140.48**
	Rosenbrock	**73601.01**	84413.69	**64690.02**	68315.59	1958799.29	**73601.56**	624341.74	**315775.79**	**188898.12**	206130.53
	Sphere	**21.15**	24.38	**20.45**	21.5	546.55	**25.67**	173.99	**96.84**	46.46	**41.03**
2	Rastrigin	84.23	**59.59**	96.73	**82.08**	**30.42**	30.76	196.99	**74.28**	157.8	**147**
	Rosenbrock	3495.21	**2386.98**	8014.1	**3814.8**	2143.16	**619.47**	73201.31	**3838.93**	17171.11	**10804.96**
	Sphere	4.25	**3.12**	8.31	**4.86**	9.98	**1.29**	68.38	**5.59**	16.55	**10.7**
3	Rastrigin	**21.06**	33.74	**24.87**	42.19	472.29	**318.94**	**247.64**	333.28	**30.98**	34.38
	Rosenbrock	**109.23**	146.53	**122.73**	159.75	1787.22	**128.55**	699754.07	**1722.27**	399.02	**307.96**
	Sphere	**0.09**	0.11	**0.09**	0.11	**0.08**	0.1	133.52	**2.03**	0.36	**0.31**
4	Rastrigin	**841.9**	871.99	772.83	**688.72**	3832	**1139.29**	**1748.45**	2848.3	754.38	**716.17**
	Rosenbrock	223395465.4	**151705461.9**	189535874	**108888429.1**	957940673.4	**205570867.9**	**477280317**	945221760	175355577.9	**127501241**
	Sphere	733.49	**609.08**	625.98	**428.99**	3541.08	**780.85**	**1431.8**	2833.45	549.18	**492.19**
$\tau = 20$											
1	Rastrigin	34.84	**33.88**	**40.16**	46.77	516.94	**483.62**	**159.09**	263.95	**46.86**	47.12
	Rosenbrock	**9210.76**	11159.41	11588.09	**10445.52**	1988246.32	**12044.2**	567717.09	**183479.33**	**19508.38**	21891.08
	Sphere	**2.86**	3.44	3.56	**3.26**	545.61	**3.46**	143.29	**43.39**	**5.4**	5.97
2	Rastrigin	20.78	**19.82**	25.47	**23.21**	42.58	**20.83**	121.64	**21.71**	46.56	**40.38**
	Rosenbrock	445.51	**290.08**	1555.23	**624.17**	105.43	**88.32**	24211.28	**1307.21**	2762.26	**1549.64**
	Sphere	0.51	**0.41**	1.38	**0.77**	12.87	**0.19**	35.24	**2.38**	2.17	**1.41**
3	Rastrigin	**11.54**	51.33	**31.99**	118.2	874.81	**802.81**	**86.23**	430.54	10.41	**8.69**
	Rosenbrock	22.3	**16.06**	21.73	**17.1**	74.6	**17.49**	204135.11	**274.62**	**11.34**	16.79
	Sphere	0.04	**0.03**	**0.03**	0.07	0.05	**0.03**	35.14	**0.52**	**0.02**	**0.02**
4	Rastrigin	129.34	**118.55**	131.28	**107.88**	3930.78	**320.03**	**717.04**	1472.18	111.38	**103.62**
	Rosenbrock	23445061.73	**16785682.65**	20298120.95	**12073567.99**	1322886269	**34616064.68**	**152205349.1**	391706791.9	16297615.87	**8864684.95**
	Sphere	74.36	**57.94**	68.51	**56.91**	4256.18	**129.66**	**588.71**	1209.84	53.91	**42.82**

8.3 Cross comparison of approaches

Figure 8.2 presents a boxplot for MOF value distribution based on each frequency for 20 runs. For this plot, we omitted the worse performing algorithms to allow for a better resolution when analysing other methods. The results demonstrates that as τ increases from 1 to 10, the MOF values decrease. For higher τ values, the algorithms have more timing budget within each change to evolve the solutions and achieve closer values to the optimum. The biggest difference in MOF values is that between $\tau = 1$ and other τ values. However, there are some exceptions to this general trend. For instance, the behaviour of NN_No in exp3 for the Rastrigin function is an anomaly. According to Figure 8.2, when increasing τ from 1 to 20, the MOF values also increase. Looking to the plot for optimum position changes in Figure 8.1, there is a linearly decreasing trend in the first half of the time scale and a constant optimum position in the second half. As NN does not have the correct new optimum (based on specific characteristics of this experiment), it is not helpful to DE. In addition, with the Rastrigin function, there are chances of getting stuck in local optima, as it has a multimodal attribute. This intensifies at higher τ values, as the population becomes more converged. However, algorithms with diversity variants can avoid the local optimum by promoting diversity. This shows the importance of diversity variants in the case of wrong predictions. NN relies on the previous time solutions achieved by DE. If the solutions are far from the optimum, the resulting training data will be poor in quality and, therefore, not longer useful to DE. Furthermore, the poor performance of NN_No (compared to its diversity variant counterparts in exp1 and exp3 for the Rastrigin function) shows how we can improve the results of NN by using diversity mechanisms. In addition, Table 8.1 shows the MOF values allowing

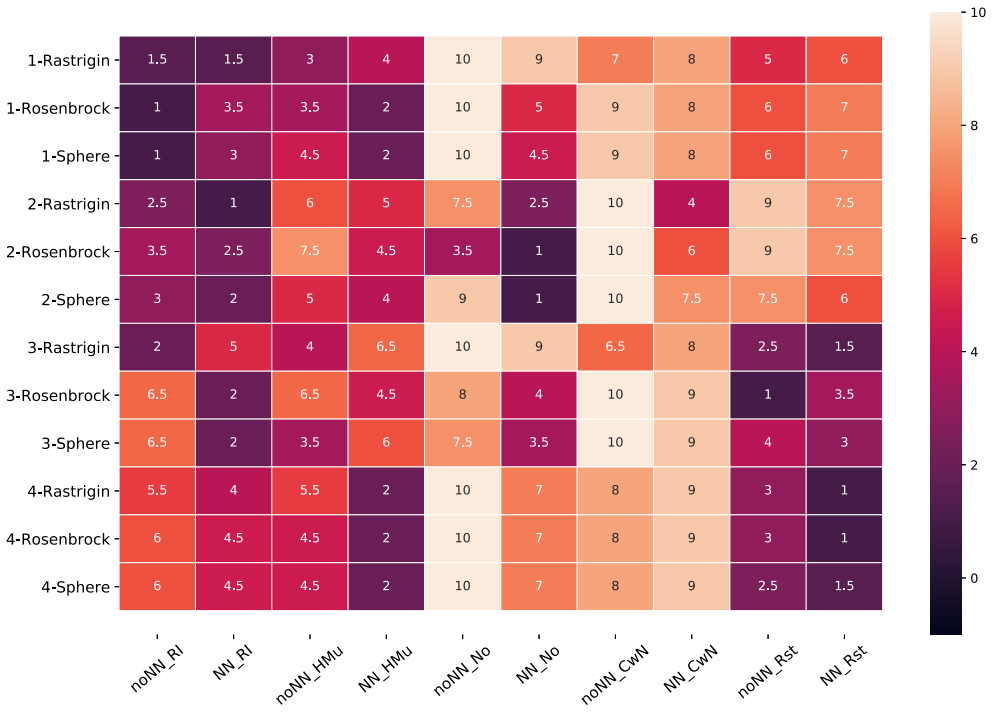


FIGURE 8.3: Heatmap of methods' rank over MOF values for each function and experiment, considering $\tau = 5$, and 10. Lower ranks represent better performances. Numbers in Y-axis label show experiment number.

pairwise comparison of diversity variants with and without NN for $\tau = 1$ and $\tau = 20$, respectively. In each set of columns, the better performing algorithm is highlighted. We clearly see that for a small τ, methods employing NN are not competitive with their counterparts in each set of diversity variants. However, for large values of τ, the trend changes and NN variants outperform noNN counterparts. First, as for this NN architecture, we train the NN with three best solutions for each time; thus, when τ is smaller, the algorithm is not converged and the best solutions have a greater distance between each other and may not represent the optimum region properly. Furthermore, for higher τ values, the NN time expenditure is negligible compared to the whole evolution process (as we see later in Table 8.2). In addition, NN gives direction to the diversity mechanisms that have a more random nature. Therefore, integration of NN and diversity variants improves the algorithm in comparison to its baseline diversity variant.

To compare the algorithms overall, a heatmap with mean rankings of MOF values is presented in Figure 8.3. To achieve these grades, we begin by ranking every method, grouping them by function, experiment and frequency. Afterwards, we calculate the mean of the ranks among all the frequencies. The heatmap helps us to analyse and compare the performance of the algorithms across each set of experiments and functions. For instance, it is clear from the heatmap that the methods using CwN are not competent in most functions

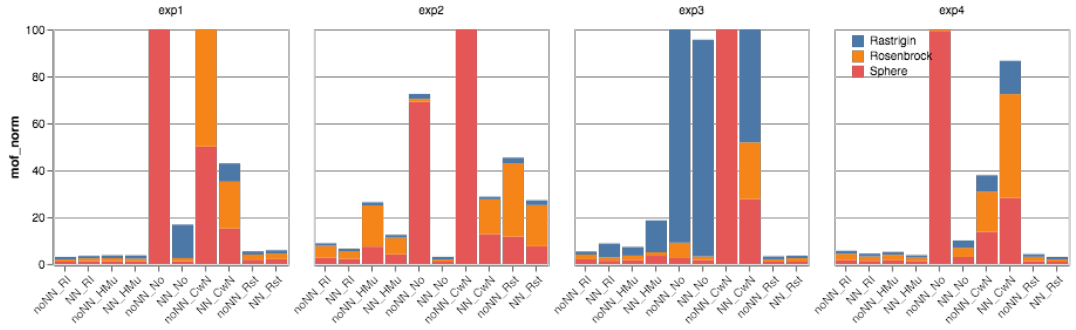


FIGURE 8.4: MOF-norm values considering $\tau = 10$ for each method and experiment, color-coded with functions

and experiments. However, this heatmap is not able to define the severity of differences among methods. For the purpose of relativity analysis, we propose Figure 8.4 which presents the results of the overall performance of methods in each experiment. To achieve standard values (denoted as MOF_norm) in each set of functions and experiments, the values are divided by the minimum value among all methods. So, for the method with the lowest MOF value, MOF_norm is assigned to one, and the other values are calculated proportionally. For example, in Figure 8.4, we can observe that CwN and Rst, in experiment 2 and 3, are considerably worse than other methods. To achieve a better resolution for the comparison of methods, we limit the y-scale, at the cost of missing some data from the worse-performing algorithms. Another observation which can be made based on this figure is that variants of RI and HMu had better performances overall.

To validate the results of the MOF values, the 95%-confidence Kruskal-Wallis statistical test and the Bonferroni post hoc test, as suggested in [37] are presented (see Figure 8.5). Nonparametric tests were adopted because the samples of runs did not fit a normal distribution, based on the Kolmogorov-Smirnov test. In this heatmap, the squares with the brightest colour show the methods with not-significantly different (NS) results, and the squares in the purple colour spectrum show the methods with statistically significant differences with the mentioned p-values. Results indicate that, in most test cases, the methods are significantly different to each other.

To compare the algorithms in each separate time look into Figure 8.6. This figure shows the deviation of the best solution achieved at each time, as well as the best_known solution for that time. Thus, it shows how the different methods track the best_known solusion, along with the changes. The values in the legend show the BEBC values for each method, that is, the average of what the plots show across all times. Lower values indicate that the methods are capable of closely following the optimum like NN_RI. Overall, most of

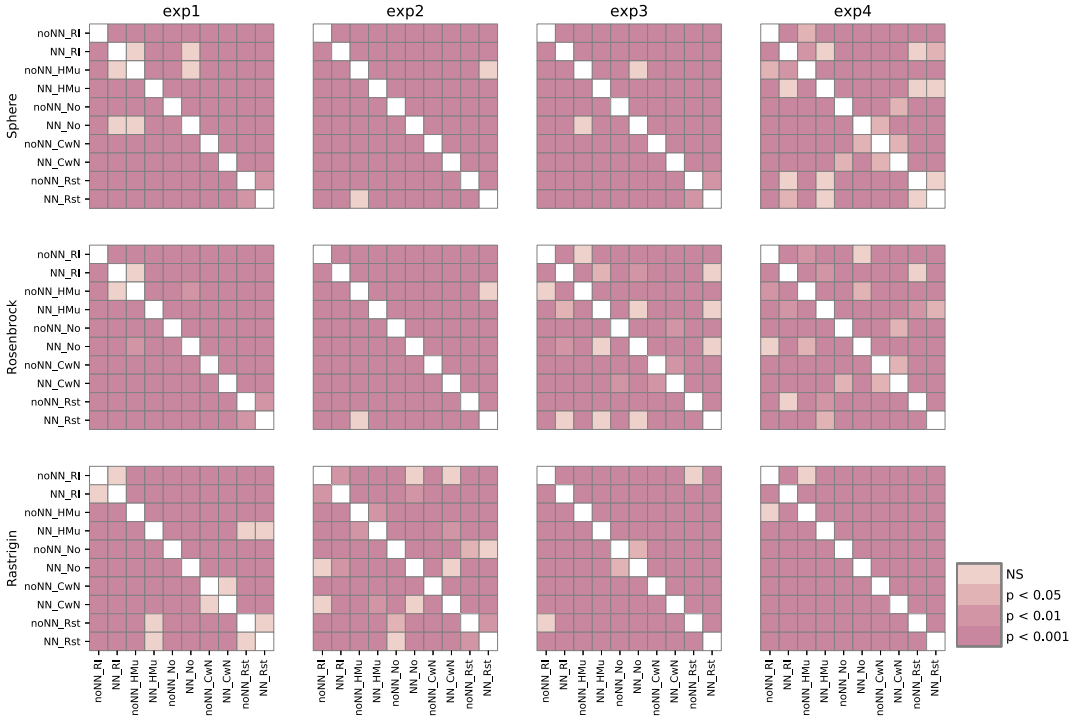


FIGURE 8.5: Kruskal-Wallis statistical test on MOF values for τ=10, NS represents not-significant

the methods show similar results; only variants of CwN and noNN_No are inferior. We only bring the results of sphere function for $\tau = 10$. However, considering other functions and frequencies, overall, when τ is small, this measure shows more differences between methods. But for $\tau = 10$ most of the methods could achieve near-to-optimum solutions regardless of the differences among their evolution processes. In addition, for the Rastrigin function with multimodal characteristics the differences between methods are bigger. This is because there are more chances of algorithms with a lack of diversity following a local optimum and becoming unable to reach near-global optimum solutions.

The other figure (Figure 8.7) shows the Euclidean distance of the position of best at the first generation after a change, and the position of best_known. This plot gives an idea of how far from the best_known position the methods start for the new time. We observe that Rst is further away as we expected, since it starts randomly. In this experiment, as the position changes are not huge (see Figure 8.1), so noNN_No is also not far away the best_known. Nevertheless, the results for this method for MOF are unsatisfactory (Table 8.1), due to the lack of diversity in the population. This lack of diversity prevents a proper exploration of the search space for a new optimum.

In Figure 8.8, the prediction error is plotted for each time. As three solutions

are sampled around the predicted solution by NN, we determined the prediction error by considering the average Euclidean distance of the best_known solution to these three inserted solutions. Notice that NN only starts to predict the future optimum position after collecting enough data, so the time scale for this plot is shorter (whereas the timescales of other plots are 100).

Figure 8.9 shows a heatmap of the algorithms' rankings, based on ARR values for every method and considering all frequencies of change. This measure shows which methods can recover more quickly after a change. ARR rank results show that the noNN version of Rst, RI and HMu variants have almost the best recovery from their first best solution after a change, compared to other methods in all experiments. As we can interpret from ARR Equation (explained in Chapter 2), this measure is slightly biased over the first solution achieved. Consequently (according to this measure), algorithms that start with a very poor solution may achieve higher ARR values than those starting with a better solution. So if the first best solution is drastically changed for the next generations, this measure reports better results (higher values). This is the reason noNN_Rst is the best based on this measure.

Conversely, the worst results for this measure are for CwN variants (both NN and noNN) and noNN_No. Comparing NN_No and noNN_No, we can conclude that NN improves the recovery capabilities of the algorithm, especially for exp1 and exp4, where we observe drastic changes.

The heatmap for SR values (see Figure 8.10) illustrates satisfactory results for almost all of the methods. This means that they can reach an ϵ-precision (=10%) of the optimum for almost all the changes (or times). However, CwN-variants and noNN_No are the exceptions in which SR values are low. In addition, all the methods showed difficulty in reaching the optimum in exp2 for the Rastrigin function (low values for SR). Moreover, in other experiments, the performance of all the methods decreased, based on SR values for this function compared to the other two functions. This is attributed to its multimodal characteristic.

Table 8.2 represents the percentages of the amount of time spent for calling NN unit compared to the overall optimisation time. Regardless of the experiment and function, the results show for $\tau = 1$ around 10-11%, $\tau = 5$ around 2%, $\tau = 10$ around 1% and $\tau = 20$ around 0.5%. This shows that when τ is higher, it is less expensive (in terms of computational costs) to use NN. As NN time remains constant, when τ is small, the proportion of time for evolution process is lower. Hence, the samples used to train NN do not represent real optimum

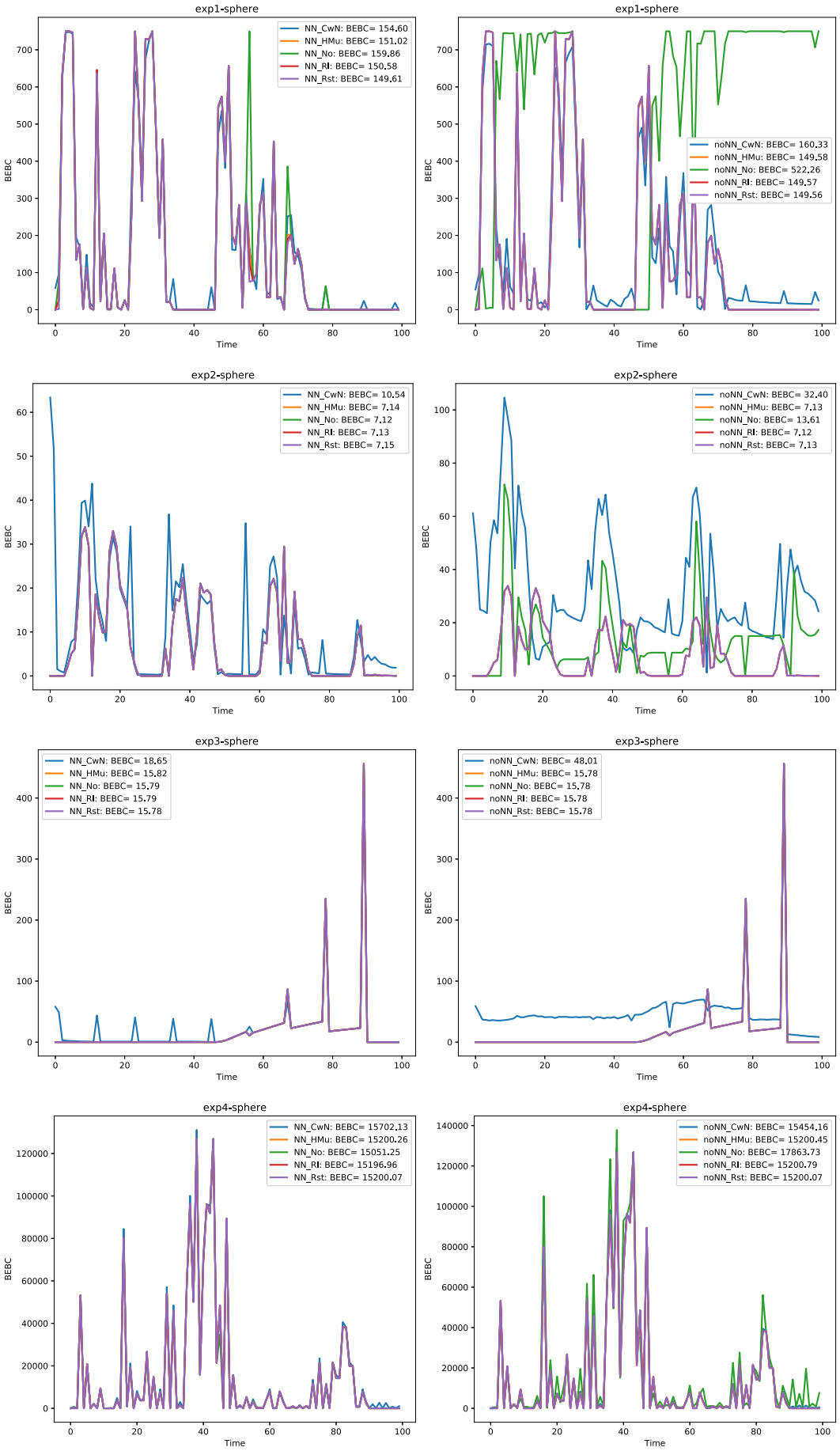


FIGURE 8.6: Best error before change values over time, $\tau = 10$.

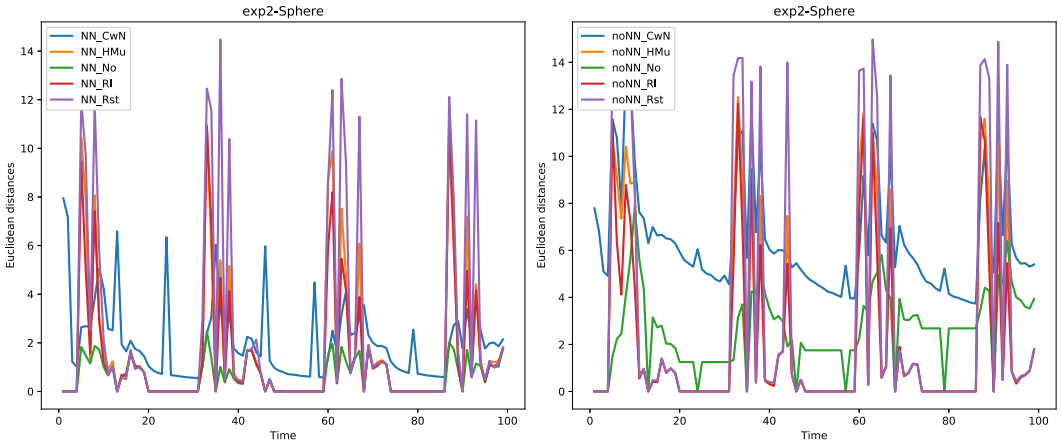


FIGURE 8.7: Euclidean distance between the best of first generation after change and the optimum position for each method, $\tau = 10$.

or near-optimum values. Consequently, predictions made using NN are not exact.

8.4 Detailed examination of the use of neural networks

In this section, the methods (NN and noNN), are compared on the basis of each diversity variant.

8.4.1 Crowding

From Figure 8.6, we observe that CwN variants exhibit the biggest difference to the optimum, in comparison to other methods. This explains their poor performance in terms of MOF values (as can be seen in Figure 8.3). Based on the colours of the heatmap, it is easily noticeable that the variants of this method perform unsatisfactorily. NN can enhance the results for this method, but it remains inferior compared to other methods. Variants of this method (NN_CwN and noNN_CwN) behave particularly poorly in exp2 and exp3, even compared to noNN_No. By promoting diversity unnecessarily, CwN adversely affects the convergence of the algorithm to new optimum position, which is not too distant from the previous optimum. In addition, from Figure 8.9, we can see that this variant delays the recovery of the algorithm (the low rank for ARR values in NN_CwN and noNN_CwN indicates the algorithm's inability to recover after a change).

Experiments of Chapter 6, regarding different diversity variants, showed that CwN has been one of the best methods for handling dynamic environments. However, CwN is not competitive in the experiments of this chapter, because

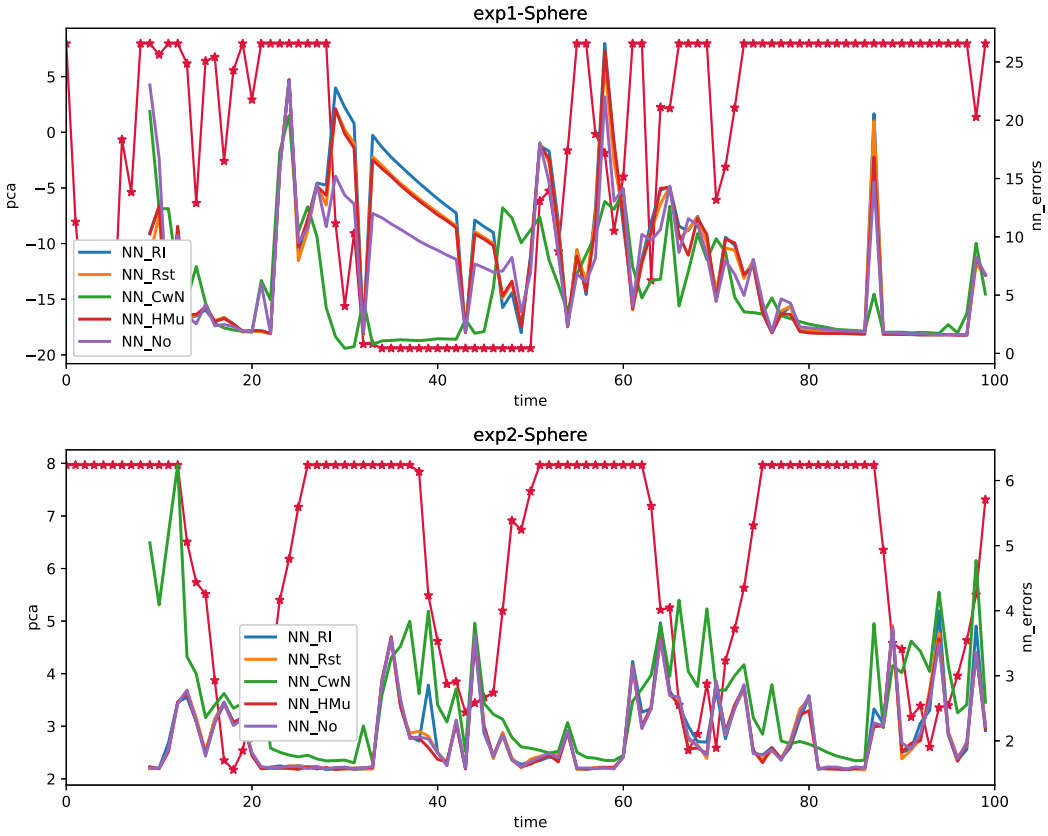


FIGURE 8.8: Error of NN plus PCA for τ=10, right y-axis shows nn_errors and left y-axis is for pca scale

of the following reasons. Firstly, in the literature, this method was most effective for multimodal test problems with several local optima. In such cases, CwN helps to diversify solutions by avoiding similar individuals in each sub-region of the search space [108]. In addition, CwN demonstrated superior performance for problems that include features such as disconnected feasible regions and small feasible areas [49]. Secondly, in experiments of Chapter 6, CwN was tested on a benchmark with a low problem dimension (2) [49]. As the problem's dimension in this chapter is 30, having a crossover rate (CR) of 0.3 leads the offspring to change in only a number of dimensions. In this condition, the closest individual to the offspring will be the parent, causing the method to act in a similar fashion to the no-diversity mechanism, but with an overhead of calculating distances at each iteration. To alleviate this in our CwN version, the offspring will compete with its five closest individuals.

Checking the results of SR-values for noNN_CwN shows this algorithm is barely able to get to the optimum for almost all the changes in exp2 for all functions and exp3 for rosenbrock function. While, as we see in Figure 8.10, most of the other methods are in 100% for exp2 (rosenbrock and sphere).

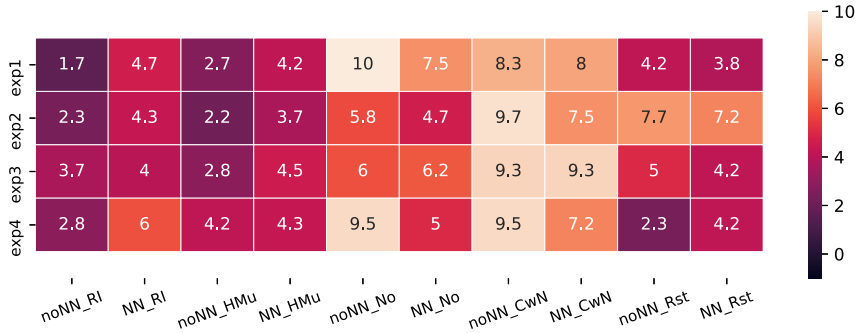


FIGURE 8.9: Heatmap of methods' rank over absolute recovery rate (ARR) values for each experiment, considering $\tau = 5$ and 10. Lower ranks represent better performances

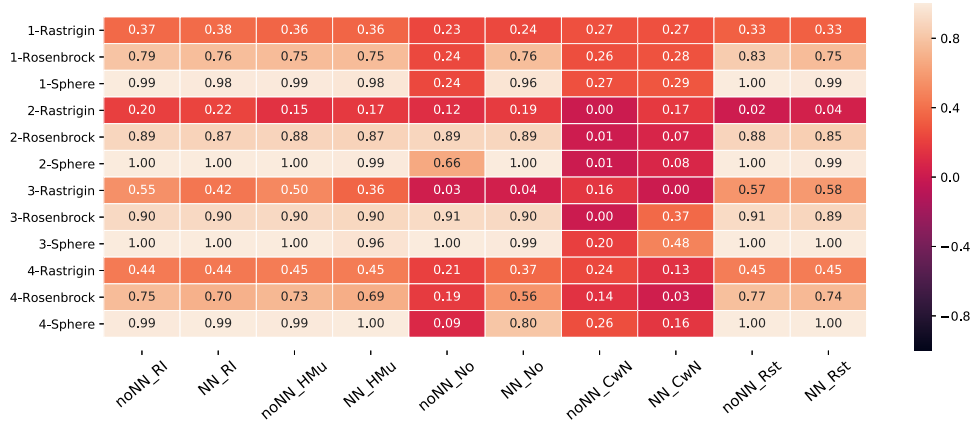


FIGURE 8.10: Heatmap of mean values (20 runs) for success rate (SR), considering $\tau = 5$ and 10. Higher values represent better performances. Numbers in Y-axis show experiments

8.4.2 Random immigrants and restart population

From Figure 8.6, it is visible that in most functions and experiments, the solutions achieved by NN_RI are the best among others in terms of tracking the optimum. This can explain its best rank amongst all methods, represented in Figure 8.3. From this figure, we can see that its noNN variant also has a promising ranking. For a clearer observation, Table 8.1 shows the comparison of NN and noNN variants for this method. Comparing them, the results for large τ for NN show better performances than noNN in most of the functions and experiments except for rastrigin with a multimodal characteristic in exp3, and rosenbrock and sphere in experiment 1 with random changes. The reason for the better performance of NN_RI is that, in noNN_RI, we only insert random solutions, while for NN_RI, we diversify solutions by random immigrants, as well as directing them with the predicted solutions by NN. This will expedite the convergence to new optimum leading to better MOF values. However, for small τ, NN has lower ability to beat its noNN counterpart. Low timing budget leave the algorithm with poor final solutions

TABLE 8.2: NN-time; % time spent for training and using NN in proportion to overall optimisation time (mean for 20 runs)

experiment		exp1				exp2				exp3				exp4			
τ		1.0	5.0	10.0	20.0	1.0	5.0	10.0	20.0	1.0	5.0	10.0	20.0	1.0	5.0	10.0	20.0
Rastrigin	NN_CwN	10.763	2.154	1.124	0.554	10.941	2.144	1.129	0.551	11.046	2.182	1.191	0.555	10.611	2.103	1.079	0.728
	NN_HMu	11.106	2.227	1.117	0.579	10.928	2.224	1.150	0.589	11.081	2.272	1.149	0.757	10.608	2.144	1.074	0.540
	NN_No	10.755	2.136	1.085	0.548	10.951	2.190	1.097	0.564	11.009	2.190	1.090	0.561	10.628	2.139	1.056	0.553
	NN_RI	11.070	2.163	1.117	0.555	10.896	2.189	1.117	0.564	11.174	2.225	1.124	0.550	10.517	2.124	1.076	0.549
	NN_Rst	10.970	2.197	1.081	0.544	10.817	2.166	1.073	0.544	11.644	2.323	1.137	0.556	10.719	2.147	1.072	0.539
Rosenbrock	NN_CwN	10.658	2.122	1.079	0.539	10.772	2.167	1.140	0.543	10.859	2.193	1.194	0.555	10.803	2.113	1.082	0.529
	NN_HMu	10.826	2.197	1.106	0.558	10.878	2.216	1.163	0.572	10.839	2.194	1.108	0.572	10.674	2.152	1.107	0.541
	NN_No	10.862	2.206	1.091	0.545	10.823	2.173	1.101	0.547	10.754	2.175	1.092	0.569	10.554	2.163	1.108	0.547
	NN_RI	10.921	2.173	1.087	0.561	10.982	2.186	1.104	0.558	11.087	2.166	1.082	0.547	10.598	2.233	1.102	0.545
	NN_Rst	11.020	2.182	1.095	0.550	10.896	2.208	1.085	0.558	11.429	2.248	1.090	0.548	11.575	2.197	1.076	0.547
Sphere	NN_CwN	10.757	2.136	1.067	0.535	10.827	2.164	1.171	0.540	10.875	2.210	1.182	0.545	10.519	2.100	1.139	0.530
	NN_HMu	10.814	2.220	1.100	0.559	10.752	2.183	1.139	0.576	10.717	2.162	1.125	0.572	10.610	2.209	1.181	0.557
	NN_No	10.952	2.211	1.128	0.560	10.830	2.988	1.098	0.539	10.682	2.170	1.088	0.552	10.643	2.152	1.074	0.540
	NN_RI	10.981	2.179	1.111	0.562	11.229	2.186	1.504	0.548	10.903	2.136	1.082	0.544	10.653	2.188	1.101	0.550
	NN_Rst	10.932	2.213	1.110	0.555	11.031	2.188	1.082	0.544	10.988	2.213	1.075	0.545	11.523	2.318	1.091	0.546

at each time. Therefore, NN is fed with poor quality solutions and thus is not able to predict the correct optimum positions for the future times. Although, worth to mention, RI is reported to have lower performance results in cases of small feasible areas [49]. The reason is the inserted solutions are discarded by constraint handling mechanism and can not proceed as best solution to guide the search for next generations. In our test problem, we lack such a small feasible area in which RI may show its worst performance. For future work, we will test this method for smaller feasible areas with features of disconnected feasible areas. In addition, another possible future work is to observe the effect of randomness over predicted solutions. In our experiments, we test with inserting five predicted solutions, as well as two random solutions. However, different combinations of these forces can be experimented based on problem features.

As mentioned before, Rst can be considered as a sever case of RI, in which, we discard all the previous attempts of the algorithm and start new random solutions. Looking to Table 8.1, if we compare NN_RI, NN_Rst and NN_No, we can conclude the following observations. In most of the experiments, we can clearly observe the behaviour of RI is significantly better than Rst method. However, noNN_Rst is ranked as the best for exp4, in which the position of the optimum changes drastically. Conversely, noNN_Rst achieves a low ranking for exp2 and exp3, in which the optimum position changes are not huge and thus by discarding the previous attempts of the algorithm, the performance degrades. This implies when using NN, diversity of population is of high importance. In case of NN_Rst, we have a population scattered around the search space. This is not helpful when NN tries to direct the solution toward the new optimum. In case of NN_RI, however, we have a proper amount of diversity among individuals of population (5 individuals from NN, and 2 individuals from RI), so the results are promising. This explains why noNN_No has an inferior performance for exp4, in which for larger changes, it can not promote diversity to reach optimum. While for

exp2 and exp3 is ranked better, since it does not need a drastic change in the position.

ARR rank results show noNN_Rst and RI variants have almost the best recovery after a change compared to other methods for all experiments. Since this measure (see ARR Equation in Chapter 2) reports better values for those algorithms that can get distant from their first generation best solution faster. For these two methods, the good result for this measure was predictable as their first best solution is randomly created and often more far away than the optimum compared to other methods.

According to SR measure, noNN_Rst show medium results. Although the performance of this algorithm drops based of MOF values for exp2 and exp3, but the results for SR values show only drop of performance in exp2 for rastrigin. This discrepancy is due to the fact that MOF values consider the performance of algorithms over all generations while SR cares only about last solution achieved by each algorithm.

8.4.3 Hyper-mutation

HMu's results are quite similar to RI variant with slightly better performances for exp1 and exp4 and worse performances for exp2 and exp3. The rankings in the heatmap 8.3, clarify this observation. The reason lies on the shape of the changes in the environment for different experiments. Exp1 and exp4 have bigger changes, in which HMu with larger scale factor after a change converge faster to the new optimum. While for exp2 and exp3 RI is more efficiently handle the smaller change. Since considering hyper mutation factor in HMu, many of the individuals go through a change and convergence is delayed. Whilst, in these two experiments the optimum position changes are minor. Due to the same reason, NN_No also outperforms NN_HMu for exp2 and exp3. This means diversity mechanisms do not always improve algorithm performance when used on top of NN. In the proposed version of HMu for DE in [5], it is proposed to change DE variant from DE/rand/1/bin to DE/best/1/bin (see Chapter 4), when hyper parameters of DE are activated. The best solution is chosen from the current time or a memory including previous times best. The test problem in that work is a two-dimensional. For larger problem sizes like ours, this method is not able to promote diversity. This is because in a DE promoting diversity needs a minimum level of diversity among solutions. To achieve this, we insert randomly created solutions (7 individuals for the case without NN and 2 individuals for the case using NN).

Although, the proper selection of the number of inserted individuals can be experimented in a future study.

Based on the statistical test results, this method does not have significant difference in some cases with RI (for exp1 sphere and rosenbrock) and in some cases with Rst (exp1 rastrigin and exp2 sphere).

8.5 Conclusions and discussions

Given the complexity of integrating NN into the evolution process, and considering the time spent to train it, we investigated whether they can be competitive solutions to solve DCOPs compared to standard diversity mechanisms. We empirically studied the possibility of integrating NN and diversity mechanisms to extract the best of each to improve the results. We observed that diversity of population is essential when using NN for DE. Because the evolution process of DE algorithm depends on the diversity of population, if we use NN without other diversity mechanism, as the inserted solutions are distributed around the predicted value, then it is slower to explore the search space leading to lower MOF values. In addition, in some cases due to the multi-modality of the function, the algorithm may get stuck in a local optimum and produce poor samples for NN to train. Hence, it is important to have a diversity mechanism. On the other hand, we observed that NN can improve the results of simple diversity mechanisms. This is because NN helps directing the search toward the next optimum as opposed to random nature of diversity mechanisms. The presented results were for one simple feed-forward NN, however, considering several proposed structures of NNs in literature, their application to handle dynamic optimisation problems is still in its infancy. So for future work, we encourage application of other NN designs to investigate their differences based of a range of problem features.

Chapter 9

Conclusions

In this thesis, evolutionary algorithms as a popular solution to tackle dynamic constrained optimisation were studied from various aspects. This class of optimisation is significantly important since many real-world problems are attributed as dynamic and constrained. For the early chapters, first we provided a background on the topic, the problem description, the state of the art algorithms, the motivation and challenges that led to this research, and the objectives of this study. In these early chapters, we also introduced benchmarks and performance metrics that were used as a reference for the technical chapters. Then, we introduced differential evolution as our chosen baseline algorithm due to its competency in continuous optimisation. Afterwards, we explained constraint and dynamic handling mechanisms that were used in the technical studies of the following chapters.

For the technical chapters (from Chapter 4 on-wards), firstly two empirical studies regarding constraint handling techniques were presented. The first one was a survey comparison over common constraint handling mechanisms, and then specifically, we elaborated in repair methods as a promising solution to handle DCOPs. Commonly proposed repair methods were applied and compared on the basis of a standard benchmark that captures different types of environmental changes. The comparisons clarified the strengths and weaknesses of each method on the basis of the problem type. In conclusion, we observed that the proper selection of constraint handling technique is more challenging where the problem is dynamic. This is because when the problem is dynamic, the dynamic handling mechanisms tend to diversify the population looking for the upcoming changes. On the other hand, constraint handling techniques tend to avoid infeasible areas directing the search toward feasible areas. We observed that repair methods can be a solution to avoid this discrepancy. As repair methods are often more relax to accept high quality (in

terms of objective values) infeasible solutions and try to direct them gradually to the feasible area.

Afterwards, we brought the empirical results of a framework that was used to create benchmarks to test algorithms in DCOPs in continuous spaces. Our proposed framework could produce multiple benchmarks to be applied for testing any function and for any number of changes and dimensions in the optimisation problem. In addition, we pointed out to the difficulty of comparing algorithms in DCOPs, and as a solution, we introduced a ranking procedure as a measure that did not need optimal points of each time for comparisons.

We continued our research with the study of diversity in evolutionary algorithms and we observed its significant importance in dynamic environments. Particularly, when the problem is constrained, examining the effect of diversity is even of a higher importance due to the mutual effects that diversity variants and constraint handling techniques would have.

For the last section of our research, we designed a neural network to be used as dynamic handling mechanism together with DE to tackle DCOPs. We proposed to consider neural network time expenditure in the experiments, as neural network may take considerable time. This is because it includes several stages: collecting data, training, and predicting, which may not be negligible in comparison to other standard mechanisms. Thus, to compare the methods fairly, we used wall clock timing as the measure for the available time between changes. In this section of our studies, the neural network parameters were calibrated running some experiments. We observed for short overall time horizons, this might not be an efficient method given that for the first change periods, we need to collect data. Moreover, training a neural network with small amounts of data will overfit the neural network, making it difficult to generalize and make predictions for new data. To alleviate this we proposed to use a couple of best solutions at each time to train the network.

We continued our investigations with the algorithm using neural network and compared it with the algorithms using common diversity mechanisms. We observed for the integration of neural network in the evolution process of DE, diversity of population has a great impact on the effectiveness of neural network. Since evolution process of DE depends on the population diversity, if we only use neural network without other diversity mechanism, as the inserted solutions are distributed around the predicted value, then it is slower to explore the search space. In addition, in some cases due to the multimodality of the function, the algorithm may get stuck in a local optimum and

produce poor samples for neural network to train. Hence, it is important to have a diversity mechanism.

For future work, considering different proposed structures for neural networks in machine learning community, there is a big capacity to test other structures combined with EAs to solve DCOPs. The other potential area to work on regarding DCOPs is to propose measurements to compare algorithms. The current measures often need optimum solutions of each time that is hard to achieve for the real-world problems. In addition, considering several features of a dynamic problem in a real-world scenario, there is still a shortage of test problems that emulate DCOPs.

Bibliography

[1] Hussein A Abbass and Kalyanmoy Deb. "Searching under multi-evolutionary pressures". In: *International Conference on Evolutionary Multi-Criterion Optimization*. Springer. 2003, pp. 391–404.

[2] M.-Y. Ameca-Alducin, E. Mezura-Montes, and N. Cruz-Ramirez. "Differential evolution with combined variants for dynamic constrained optimization". In: *Evolutionary Computation (CEC), 2014 IEEE Congress on*. 2014, pp. 975–982.

[3] Maria-Yaneli Ameca-Alducin, Maryam Hasani-Shoreh, and Frank Neumann. "On the use of repair methods in differential evolution for dynamic constrained optimization". In: *International Conference on the Applications of Evolutionary Computation*. Springer. 2018, pp. 832–847.

[4] María-Yaneli Ameca-Alducin, Efrén Mezura-Montes, and Nicandro Cruz-Ramírez. "A repair method for differential evolution with combined variants to solve dynamic constrained optimization problems". In: *Proceedings of the 2015 annual conference on genetic and evolutionary computation*. 2015, pp. 241–248.

[5] Maria-Yaneli Ameca-Alducin, Efrén Mezura-Montes, and Nicandro Cruz-Ramirez. "Differential evolution with combined variants for dynamic constrained optimization". In: *Evolutionary computation (CEC), 2014 IEEE congress on*. 2014, pp. 975–982.

[6] María-Yaneli Ameca-Alducin, Efrén Mezura-Montes, and Nicandro Cruz-Ramírez. "Differential Evolution with Combined Variants plus a Repair Method to Solve Dynamic Constrained Optimization Problems: A comparative study". In: *Soft Computing* (2016), pp. 1–30.

[7] María-Yaneli Ameca-Alducin et al. "A comparison of constraint handling techniques for dynamic constrained optimization problems". In: *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2018, pp. 1–8.

[8] HC Andersen. "An investigation into genetic algorithms, and the relationship between speciation and the tracking of optima in dynamic functions". In: *Brisbane, Australia: Honors, Queensland Univ* (1991).

[9] Aniruddha Basak, Swagatam Das, and Kay Chen Tan. "Multimodal optimization using a biobjective differential evolution algorithm enhanced with mean distance-based selection". In: *IEEE Transactions on Evolutionary Computation* 17.5 (2013), pp. 666–685.

[10] Tim Blackwell and Jürgen Branke. "Multiswarms, exclusion, and anti-convergence in dynamic environments". In: *IEEE transactions on evolutionary computation* 10.4 (2006), pp. 459–472.

[11] Tim M Blackwell and Peter J Bentley. "Dynamic search with charged swarms". In: *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. Citeseer. 2002, pp. 19–26.

[12] Peter AN Bosman. "Learning, anticipation and time-deception in evolutionary online dynamic optimization". In: *Proceedings of the 7th annual workshop on Genetic and evolutionary computation*. 2005, pp. 39–47.

[13] Peter AN Bosman and Han La Poutre. "Learning and anticipation in online dynamic optimization with evolutionary algorithms: the stochastic case". In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM. 2007, pp. 1165–1172.

[14] J. Branke. "Memory enhanced evolutionary algorithms for changing optimization problems". In: *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*. Vol. 3. 1999, 1882 Vol. 3.

[15] Jürgen Branke. *Evolutionary optimization in dynamic environments*. Vol. 3. Springer Science & Business Media, 2012.

[16] Jürgen Branke and Hartmut Schmeck. "Designing evolutionary algorithms for dynamic optimization problems". In: *Advances in evolutionary computing*. Springer, 2003, pp. 239–262.

[17] Jürgen Branke et al. "A multi-population approach to dynamic optimization problems". In: *Evolutionary design and manufacture*. Springer, 2000, pp. 299–307.

[18] C. Bu, W. Luo, and L. Yue. "Continuous Dynamic Constrained Optimization with Ensemble of Locating and Tracking Feasible Regions Strategies". In: *IEEE Transactions on Evolutionary Computation* PP.99 (2016), pp. 1–1.

[19] Chenyang Bu, Wenjian Luo, and Lihua Yue. "Continuous dynamic constrained optimization with ensemble of locating and tracking feasible regions strategies". In: *IEEE Transactions on Evolutionary Computation* 21.1 (2017), pp. 14–33.

[20] Chenyang Bu, Wenjian Luo, and Tao Zhu. "Differential evolution with a species-based repair strategy for constrained optimization". In: *Evolutionary Computation (CEC), 2014 IEEE Congress on*. 2014, pp. 967–974.

[21] L. T. Bui, H. A. Abbass, and J. Branke. "Multiobjective optimization for dynamic environments". In: *2005 IEEE Congress on Evolutionary Computation*. Vol. 3. 2005, 2349–2356 Vol. 3.

[22] Walter Cedeno and V Rao Vemuri. "On the use of niching for dynamic landscapes". In: *Proceedings of 1997 Ieee International Conference on Evolutionary Computation (Icec'97)*. IEEE. 1997, pp. 361–366.

[23] Piya Chootinan and Anthony Chen. "Constraint handling in genetic algorithms using a gradient-based repair method". In: *Computers & operations research* 33.8 (2006), pp. 2263–2281.

[24] H. Cobb. *An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-Dependent Nonstationary Environments*. Tech. rep. Naval Research Lab Washington DC, 1990.

[25] Helen G. Cobb. "An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-Dependent Nonstationary Environments". In: 1990.

[26] Helen G Cobb. *An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments*. Tech. rep. Naval Research lab Washington DC, 1990.

[27] Emma Collingwood, David Corne, and Peter Ross. "Useful diversity via multiploidy". In: *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE. 1996, pp. 810–813.

[28] L. Contreras-Varela and E. Mezura-Montes. "A Diversity Promotion Study in Constrained Optimizations". In: *2018 IEEE Congress on Evolutionary Computation (CEC)*. 2018, pp. 1–8.

[29] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. "Exploration and exploitation in evolutionary algorithms: A survey". In: *ACM Computing Surveys (CSUR)* 45.3 (2013), p. 35.

[30] Moayed Daneshyari and Gary G Yen. "Dynamic optimization using cultural based PSO". In: *2011 IEEE Congress of Evolutionary Computation (CEC)*. IEEE. 2011, pp. 509–516.

[31] Swagatam Das, Sankha Subhra Mullick, and Ponnuthurai N Suganthan. "Recent advances in differential evolution–an updated survey". In: *Swarm and Evolutionary Computation* 27 (2016), pp. 1–30.

[32] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. "Differential evolution: A survey of the state-of-the-art". In: *IEEE transactions on evolutionary computation* 15.1 (2011), pp. 4–31.

[33] Fabrício Olivetti De França and Fernando J Von Zuben. "A dynamic artificial immune algorithm applied to challenging benchmarking problems". In: *2009 IEEE Congress on Evolutionary Computation*. IEEE. 2009, pp. 423–430.

[34] C De Prada et al. "Plant-wide control of a hybrid process". In: *International Journal of Adaptive Control and Signal Processing* 22.2 (2008), pp. 124–141.

[35] Kalyanmoy Deb. "An efficient constraint handling method for genetic algorithms". In: *Computer methods in applied mechanics and engineering* 186.2 (2000), pp. 311–338.

[36] Kalyanmoy Deb, S Karthik, et al. "Dynamic multi-objective optimization and decision-making using modified NSGA-II: a case study on hydro-thermal power scheduling". In: *International conference on evolutionary multi-criterion optimization*. Springer. 2007, pp. 803–817.

[37] Joaquín Derrac et al. "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms". In: *Swarm and Evolutionary Computation* 1.1 (2011), pp. 3–18.

[38] Sébastien Le Digabel and Stefan M Wild. "A taxonomy of constraints in simulation-based optimization". In: *arXiv preprint arXiv:1505.07881* (2015).

[39] Jeroen Eggermont et al. "Raising the dead: Extending evolutionary algorithms with a case-based memory". In: *European Conference on Genetic Programming*. Springer. 2001, pp. 280–290.

[40] Saber M Elsayed, Tapabrata Ray, and Ruhul A Sarker. "A surrogate-assisted differential evolution algorithm with dynamic parameters selection for solving expensive optimization problems". In: *Evolutionary Computation (CEC), 2014 IEEE Congress on*. 2014, pp. 1062–1068.

[41] Jose Luis Fernandez-Marquez and Josep Lluis Arcos. "Adapting particle swarm optimization in dynamic and noisy environments". In: *IEEE Congress on Evolutionary Computation*. IEEE. 2010, pp. 1–8.

[42] Patryk Filipiak and Piotr Lipinski. "Dynamic portfolio optimization in ultra-high frequency environment". In: *European Conference on the Applications of Evolutionary Computation*. Springer. 2017, pp. 34–50.

[43] Wanru Gao and Frank Neumann. "Runtime analysis for maximizing population diversity in single-objective optimization". In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2014, pp. 777–784.

[44] Ashish Ghosh and Shigeyoshi Tsutsui. *Advances in evolutionary computing: theory and applications*. Springer Science & Business Media, 2012.

[45] C. K. Goh and K. C. Tan. "A Competitive-Cooperative Coevolutionary Paradigm for Dynamic Multiobjective Optimization". In: *IEEE Transactions on Evolutionary Computation* 13.1 (2009), pp. 103–127.

[46] John J Grefenstette et al. "Genetic algorithms for changing environments". In: *PPSN*. Vol. 2. 1992, pp. 137–144.

[47] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)". In: *Evolutionary computation* 11.1 (2003), pp. 1–18.

[48] Maryam Hasani-Shoreh, Renato Hermoza Aragonés, and Frank Neumann. "Neural Networks in Evolutionary Dynamic Constrained Optimization: Computational Cost and Benefits". In: 24th European Conference on Artificial Intelligence 325 (2020), pp. 275–282.

[49] Maryam Hasani-Shoreh and Frank Neumann. "On the Use of Diversity Mechanisms in Dynamic Constrained Continuous Optimization". In: *International Conference on Neural Information Processing*. Springer. 2019, pp. 644–657.

[50] Maryam Hasani-Shoreh et al. "On the behaviour of differential evolution for problems with dynamic linear constraints". In: *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2019, pp. 3045–3052.

[51] Iason Hatzakis and David Wallace. "Dynamic multi-objective optimization with evolutionary algorithms: a forward-looking approach". In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. 2006, pp. 1201–1208.

[52] Xiaohui Hu and Russell C Eberhart. "Adaptive particle swarm optimization: detection and response to dynamic systems". In: *Proceedings of the 2002 Congress on Evolutionary Computation.* Vol. 2. IEEE. 2002, pp. 1666–1670.

[53] Marcus Hutter and Shane Legg. "Fitness uniform optimization". In: *IEEE Transactions on Evolutionary Computation* 10.5 (2006), pp. 568–589.

[54] Hisao Ishibuchi, Noritaka Tsukamoto, and Yusuke Nojima. "Diversity improvement by non-geometric binary crossover in evolutionary multiobjective optimization". In: *IEEE Transactions on Evolutionary Computation* 14.6 (2010), pp. 985–998.

[55] Stefan Janson and Martin Middendorf. "A hierarchical particle swarm optimizer and its adaptive variant". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 35.6 (2005), pp. 1272–1282.

[56] Stefan Janson and Martin Middendorf. "A hierarchical particle swarm optimizer for noisy and dynamic environments". In: *Genetic Programming and Evolvable Machines* 7.4 (2006), pp. 329–354.

[57] Dongli Jia, Guoxin Zheng, and Muhammad Khurram Khan. "An effective memetic differential evolution algorithm based on chaotic local search". In: *Information Sciences* 181.15 (2011), pp. 3175–3187.

[58] Min Jiang et al. "Transfer learning-based dynamic multiobjective optimization algorithms". In: *IEEE Transactions on Evolutionary Computation* 22.4 (2017), pp. 501–514.

[59] Shouyong Jiang and Shengxiang Yang. "Evolutionary Dynamic Multiobjective Optimization: Benchmarks and Algorithm Comparisons." In: *IEEE Trans. Cybernetics* 47.1 (2017), pp. 198–211.

[60] Yaochu Jin and Bernhard Sendhoff. "Constructing dynamic optimization test problems using the multi-objective optimization concept". In: *Workshops on Applications of Evolutionary Computation*. Springer. 2004, pp. 525–536.

[61] Oliver Kramer. *A brief introduction to continuous evolutionary optimization*. Springer, 2014.

[62] Changhe Li and Shengxiang Yang. "A clustering particle swarm optimizer for dynamic optimization". In: *2009 IEEE congress on evolutionary computation*. IEEE. 2009, pp. 439–446.

[63] Changhe Li et al. *Benchmark generator for CEC 2009 competition on dynamic optimization*. Tech. rep. 2008.

[64] Xiaodong Li, Jürgen Branke, and Tim Blackwell. "Particle swarm with speciation and adaptation in a dynamic environment". In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. 2006, pp. 51–58.

[65] J. J. Liang et al. *Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-Parameter Optimization*. Tech. rep. Singapure: Nanyang Technological University, Singapore, 2005.

[66] JJ Liang et al. "Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization". In: *Journal of Applied Mechanics* 41.8 (2006), pp. 8–31.

[67] Li Liu et al. "Adaptive contamination source identification in water distribution systems using an evolutionary algorithm-based dynamic optimization procedure". In: *Water Distribution Systems Analysis Symposium 2006*. 2008, pp. 1–9.

[68] Lili Liu, Dingwei Wang, and Shengxiang Yang. "Compound particle swarm optimization in dynamic environments". In: *Workshops on Applications of Evolutionary Computation*. Springer. 2008, pp. 616–625.

[69] Xiao-Fang Liu, Zhi-Hui Zhan, and Jun Zhang. "Neural network for change direction prediction in dynamic optimization". In: *IEEE Access* 6 (2018), pp. 72649–72662.

[70] Xiao-Fang Liu et al. "Neural Network-Based Information Transfer for Dynamic Optimization". In: *IEEE transactions on neural networks and learning systems* (2019).

[71] Rodica Ioana Lung and D Dumitrescu. "A new collaborative evolutionary-swarm optimization technique". In: *Proceedings of the 9th annual conference companion on Genetic and evolutionary computation*. 2007, pp. 2817–2820.

[72] Almuth Meier and Oliver Kramer. "Prediction with recurrent neural networks in evolutionary dynamic optimization". In: *International Conference on the Applications of Evolutionary Computation*. Springer. 2018, pp. 848–863.

[73] Almuth Meier and Oliver Kramer. "Predictive Uncertainty Estimation with Temporal Convolutional Networks for Dynamic Evolutionary Optimization". In: *International Conference on Artificial Neural Networks*. Springer. 2019, pp. 409–421.

[74] R. Mendes and A.S. Mohais. "DynDE: a differential evolution for dynamic optimization problems". In: *Evolutionary Computation, 2005. The 2005 IEEE Congress on*. Vol. 3. 2005, 2808–2815 Vol. 3.

[75] Koenraad Mertens, Tom Holvoet, and Yolande Berbers. "The DynCOAA algorithm for dynamic constraint optimization problems". In: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. ACM. 2006, pp. 1421–1423.

[76] Efrén Mezura-Montes and Carlos A Coello Coello. "A simple multimembered evolution strategy to solve constrained optimization problems". In: *IEEE Transactions on Evolutionary computation* 9.1 (2005), pp. 1–17.

[77] Efrén Mezura-Montes and Carlos A Coello Coello. "Constraint-handling in nature-inspired numerical optimization: past, present and future". In: *Swarm and Evolutionary Computation* 1.4 (2011), pp. 173–194.

[78] Efrén Mezura-Montes, Carlos A Coello Coello, and Edy I Tun-Morales. "Simple feasibility rules and differential evolution for constrained optimization". In: *Mexican International Conference on Artificial Intelligence*. Springer. 2004, pp. 707–716.

[79] Z. Michalewicz and G. Nazhiyath. "Genocop III: a co-evolutionary algorithm for numerical optimization problems with nonlinear constraints". In: *Evolutionary Computation, 1995., IEEE International Conference on*. Vol. 2. 1995, pp. 647–651.

[80] Zbigniew Michalewicz and David B Fogel. *How to solve it: modern heuristics*. Springer Science & Business Media, 2013.

[81] Juan M Morales et al. "Managing Uncertainty with Flexibility". In: *Integrating Renewables in Electricity Markets*. Springer, 2014, pp. 137–171.

[82] Naoki Mori, Hajime Kita, and Yoshikazu Nishikawa. "Adaptation to a changing environment by means of the thermodynamical genetic algorithm". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 1996, pp. 513–522.

[83] Ronald W Morrison. *Designing evolutionary algorithms for dynamic environments*. Springer Science & Business Media, 2013.

[84] Khim Peow Ng and Kok Cheong Wong. "A new diploid scheme and dominance change mechanism for non-stationary function optimization". In: *Proceedings of the 6th international conference on genetic algorithms*. 1995, pp. 159–166.

[85] T. T. Nguyen and X. Yao. "Benchmarking and solving dynamic constrained problems". In: *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*. 2009, pp. 690–697.

[86] Trung Thanh Nguyen, Shengxiang Yang, and Juergen Branke. "Evolutionary dynamic optimization: A survey of the state of the art". In: *Swarm and Evolutionary Computation* 6 (2012), pp. 1–24.

[87] Trung Thanh Nguyen and Xin Yao. "Continuous dynamic constrained optimization—The challenges". In: *IEEE Transactions on Evolutionary Computation* 16.6 (2012), pp. 769–786.

[88] Trung Thanh Nguyen and Xin Yao. "Detailed experimental results of GA, RIGA, HyperM and GA+ Repair on the G24 set of benchmark problems". In: *School of Computer Science, University of Birmingham, Tech. Rep* (2010).

[89] Trung Thanh Nguyen and Xin Yao. "Solving dynamic constrained optimisation problems using repair methods". In: *IEEE Transactions on Evolutionary Computation (submitted)* (2010).

[90] T.T. Nguyen. *A proposed real-valued dynamic constrained benchmark set*. Tech. rep. School Comput. Sci., Univ. Birmingham, Birmingham, U.K., 2008.

[91] T.T. Nguyen. "Continuous Dynamic Optimisation Using Evolutionary Algorithms". PhD thesis. School of Computer Science The University of Birmingham, 2010.

[92] Mohammad Nabi Omidvar et al. "Cooperative co-evolution with differential grouping for large scale optimization". In: *IEEE Transactions on Evolutionary Computation* 18.3 (2014), pp. 378–393.

[93] Franz Oppacher, Mark Wineberg, et al. "The shifting balance genetic algorithm: Improving the GA in a dynamic environment". In: *Proceedings of the genetic and evolutionary computation conference*. Vol. 1. 1999, pp. 504–510.

[94] K. Pal et al. "Dynamic Constrained Optimization with offspring repair based Gravitational Search Algorithm". In: *Evolutionary Computation (CEC), 2013 IEEE Congress on*. 2013, pp. 2414–2421.

[95] In: *Swarm, Evolutionary, and Memetic Computing*. Ed. by BijayaKetan Panigrahi et al. Vol. 8297. Lecture Notes in Computer Science. 2013. ISBN: 978-3-319-03752-3.

[96] Diego Martinez Prata, Enrique Luis Lima, and José Carlos Pinto. "Simultaneous data reconciliation and parameter estimation in bulk polypropylene polymerizations in real time". In: *Macromolecular Symposia*. Vol. 243. 1. Wiley Online Library. 2006, pp. 91–103.

[97] Shahryar Rahnamayan, Hamid R Tizhoosh, and Magdy MA Salama. "Opposition-based differential evolution". In: *IEEE Transactions on Evolutionary computation* 12.1 (2008), pp. 64–79.

[98] Pratyusha Rakshit et al. "Uncertainty management in differential evolution induced multiobjective optimization in presence of measurement noise". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 44.7 (2014), pp. 922–937.

[99] Hendrik Richter. "Evolutionary Computation for Dynamic Optimization Problems". In: Springer Berlin Heidelberg, 2013. Chap. Dynamic Fitness Landscape Analysis, pp. 269–297.

[100] Hendrik Richter. "Memory design for constrained dynamic optimization problems". In: *European Conference on the Applications of Evolutionary Computation*. Springer. 2010, pp. 552–561.

[101] Hendrik Richter and Shengxiang Yang. "Learning behavior in abstract memory schemes for dynamic optimization problems". In: *Soft Computing* 13.12 (2009), pp. 1163–1173.

[102] Hendrik Richter and Shengxiang Yang. "Memory based on abstraction for dynamic fitness functions". In: *Workshops on Applications of Evolutionary Computation*. Springer. 2008, pp. 596–605.

[103] Marius Riekert, KM Malan, and AP Engelbrect. "Adaptive genetic programming for dynamic classification problems". In: *2009 IEEE Congress on Evolutionary Computation*. IEEE. 2009, pp. 674–681.

[104] Philipp Rohlfshagen, Per Kristian Lehre, and Xin Yao. "Dynamic evolutionary optimisation: an analysis of frequency and magnitude of change". In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM. 2009, pp. 1713–1720.

[105] Vahid Roostapour, Aneta Neumann, and Frank Neumann. "On the performance of baseline evolutionary algorithms on the dynamic knapsack problem". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2018, pp. 158–169.

[106] Claudio Rossi, Mohamed Abderrahim, and Julio César Díaz. "Tracking moving optima using Kalman-based predictions". In: *Evolutionary computation* 16.1 (2008), pp. 1–30.

[107] Thomas P. Runarsson and Xin Yao. "Stochastic ranking for constrained evolutionary optimization". In: *IEEE Transactions on evolutionary computation* 4.3 (2000), pp. 284–294.

[108] Bruno Sareni and Laurent Krahenbuhl. "Fitness sharing and niching methods revisited". In: *IEEE transactions on Evolutionary Computation* 2.3 (1998), pp. 97–106.

[109] Anabela Simões and Ernesto Costa. "An immune system-based genetic algorithm to deal with dynamic environments: diversity and memory". In: *Artificial Neural Nets and Genetic Algorithms*. Springer. 2003, pp. 168–174.

[110] Anabela Simões and Ernesto Costa. "Evolutionary algorithms for dynamic environments: prediction using linear regression and markov chains". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2008, pp. 306–315.

[111] Anabela Simões and Ernesto Costa. "Improving memory's usage in evolutionary algorithms for changing environments". In: *2007 IEEE Congress on Evolutionary Computation*. IEEE. 2007, pp. 276–283.

[112] Anabela Simões and Ernesto Costa. "Improving prediction in evolutionary algorithms for dynamic environments". In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM. 2009, pp. 875–882.

[113] Dirk Sudholt. "The benefits of population diversity in evolutionary algorithms: a survey of rigorous runtime analyses". In: *Theory of Evolutionary Computation*. Springer, 2020, pp. 359–404.

[114] Tetsuyuki Takahama, Setsuko Sakai, and Noriyuki Iwane. "Constrained optimization by the epsilon constrained hybrid algorithm of particle swarm optimization and genetic algorithm". In: *Australian Conference on Artificial Intelligence*. Vol. 3809. Springer. 2005, pp. 389–400.

[115] Ke Tang et al. "Benchmark functions for the CEC'2008 special session and competition on large scale global optimization". In: *Nature inspired computation and applications laboratory, USTC, China* 24 (2007), pp. 1–18.

[116] Biruk Tessema and Gary G Yen. "An adaptive penalty formulation for constrained evolutionary optimization". In: *IEEE Transactions on*

Systems, Man, and Cybernetics-Part A: Systems and Humans 39.3 (2009), pp. 565–578.

[117] Andrea Toffolo and Ernesto Benini. "Genetic diversity as an objective in multi-objective evolutionary algorithms". In: *Evolutionary computation* 11.2 (2003), pp. 151–167.

[118] Shigeyoshi Tsutsui, Yoshiji Fujimoto, and Ashish Ghosh. "Forking genetic algorithms: GAs with search space division schemes". In: *Evolutionary computation* 5.1 (1997), pp. 61–80.

[119] Rasmus K Ursem et al. "Multinational GAs: Multimodal Optimization Techniques in Dynamic Environments." In: *GECCO*. Vol. 20. 0. 2000, p. 0.

[120] 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*. 1997, pp. 355–360.

[121] Frank Vavak, Terence C Fogarty, and Ken Jukes. "A genetic algorithm with variable range of local search for tracking changing environments". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 1996, pp. 376–385.

[122] Yao Wang and Mark Wineberg. "Estimation of evolvability genetic algorithm and dynamic environments". In: *Genetic Programming and Evolvable Machines* 7.4 (2006), p. 355.

[123] Y. G. Woldesenbet and G. G. Yen. "Dynamic Evolutionary Algorithm With Variable Relocation". In: *IEEE Transactions on Evolutionary Computation* 13.3 (2009), pp. 500–513.

[124] Yonas G Woldesenbet and Gary G Yen. "Dynamic evolutionary algorithm with variable relocation". In: *IEEE Transactions on Evolutionary Computation* 13.3 (2009), pp. 500–513.

[125] S. Yang and X. Yao. "Population-Based Incremental Learning With Associative Memory for Dynamic Environments". In: *IEEE Transactions on Evolutionary Computation* 12.5 (2008), pp. 542–561.

[126] Shengxiang Yang. "A comparative study of immune system based genetic algorithms in dynamic environments". In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. 2006, pp. 1377–1384.

[127] Shengxiang Yang. "Associative memory scheme for genetic algorithms in dynamic environments". In: *Workshops on Applications of Evolutionary Computation*. Springer. 2006, pp. 788–799.

[128] Shengxiang Yang. "Genetic algorithms with memory-and elitism-based immigrants in dynamic environments". In: *Evolutionary Computation* 16.3 (2008), pp. 385–416.

[129] Shengxiang Yang. "Memory-based immigrants for genetic algorithms in dynamic environments". In: *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. 2005, pp. 1115–1122.

[130] Shengxiang Yang. "On the design of diploid genetic algorithms for problem optimization in dynamic environments". In: *IEEE International Conference on Evolutionary Computation*. 2006, pp. 1362–1369.

[131] Shengxiang Yang and Changhe Li. "A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments". In: *IEEE Transactions on Evolutionary Computation* 14.6 (2010), pp. 959–974.

[132] Shengxiang Yang and Xin Yao. "Experimental study on population-based incremental learning algorithms for dynamic optimization problems". In: *Soft computing* 9.11 (2005), pp. 815–834.

[133] Xiaodong Yin and Noel Germay. "A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization". In: *Artificial neural nets and genetic algorithms*. Springer. 1993, pp. 450–457.

[134] EL Yu and Ponnuthurai N Suganthan. "Evolutionary programming with ensemble of explicit memories for dynamic optimization". In: *2009 IEEE Congress on Evolutionary Computation*. IEEE. 2009, pp. 431–438.

[135] Zhuhong Zhang et al. "Danger theory based artificial immune system solving dynamic constrained single-objective optimization". In: *Soft Computing* 18.1 (2014), pp. 185–206.

[136] Aimin Zhou, Yaochu Jin, and Qingfu Zhang. "A population prediction strategy for evolutionary dynamic multiobjective optimization". In: *IEEE transactions on cybernetics* 44.1 (2013), pp. 40–53.

[137] Aimin Zhou et al. "Prediction-based population re-initialization for evolutionary dynamic multi-objective optimization". In: *International Conference on Evolutionary Multi-Criterion Optimization*. Springer. 2007, pp. 832–846.

[138] Tao Zhu, Wenjian Luo, and Zhifang Li. "An adaptive strategy for updating the memory in evolutionary algorithms for dynamic optimization". In: *2011 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE)*. IEEE. 2011, pp. 8–15.

[139] Albert Y. Zomaya and Yee-Hwei Teh. "Observations on using genetic algorithms for dynamic load-balancing". In: *IEEE transactions on parallel and distributed systems* 12.9 (2001), pp. 899–911.