MULTI-SPECIES EVOLUTIONARY ALGORITHMS FOR COMPLEX OPTIMISATION PROBLEMS

by

XIAOFEN LU

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

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
November 2018

ABSTRACT

Evolutionary algorithms (EAs) face challenges when meeting optimisation problems that are large-scale, multi-disciplinary, or dynamic, etc. To address the challenges, this thesis focuses on developing specific and efficient multi-species EAs to deal with concurrent engineering (CE) problems and dynamic constrained optimisation problems (DCOPs). The main contributions of this thesis are:

First, to achieve a better collaboration among different sub-problem optimisation, it proposes two novel collaboration strategies when using cooperative co-evolution to solve two typical kinds of CE problems. Both help to obtain designs of higher quality. An effective method is also given to adjust the communication frequency among different sub-problem optimisation.

Second, it develops a novel dynamic handling strategy for DCOPs, which applies speciation methods to maintain individuals in different feasible regions. Experimental studies show that it generally reacts faster than the state-of-the-art algorithms on a test set of DCOPs.

Third, it proposes another novel dynamic handling strategy based on competitive co-evolution (ComC) to address fast-changing DCOPs. It employs ComC to find a promising solution set beforehand and uses it for initialisation when detecting a change. It is shown by experiments that this strategy can help adapt to environmental changes well especially for DCOPs with very fast changes.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my supervisor Prof. Xin Yao. This thesis would not have been completed without his patient supervision, valuable guidance and substantial support. During my PhD study, his enthusiasm in research, optimism, scrupulous attention to details, and discipline have inspired me a lot. He is always a role model of scholar to me. Thank him very much for the encouragement before my PhD viva.

The second thanks are given to my two co-supervisors, Prof. Ke Tang from the SUSTech and Dr. Stefan Menzel from the Honda Research Institute Europe, both of whom offered me many valuable comments and insightful suggestions on both my research and future career. This thesis would not have been completed without their careful guidance. I am also greatly indebted to the Honda Research Institute Europe for funding my PhD study.

Special thanks to my thesis group members, Prof. John Barnden (RSMG representative) and Dr. Ata Kaban, who have given helpful comments and valuable suggestions in every thesis group meeting. All of these have greatly improved the thesis and my way of doing research. Thank Dr. Kaban again for being the internal examiner for my thesis and arranging my PhD viva. I really appreciate the valuable correction advice from her.

I would also like to thank Prof. Shengxiang Yang for being the external examiner for my thesis and Prof. Andrew Howes for being the chairman for my PhD viva. Thank Prof. Yang again for all the carefully marked corrections.

Luckily, I have met some excellent colleagues, senior researchers and friends during my PhD study. Thank Dr. Haobo Fu, Ruiwen Zhang, Dr. Weiqi Chen, Dr Di Zhang,

Liyan Song, Dr. Fengzhen Tang, Dr. Renzhi Chen, Dr. Guanbo Jia, Dr. Shuo Wang, Dr. Leandro Minku, Mirabella, Michael, Dr. Mang Wang, Pietro, Hanno, Esra, Momodou, Xu Wang, Dr. Dongsheng He, Dr. Miqing Li, Dr. Ke Li, Dr. Ran Cheng, Prof. Guofu Zhang, Prof. Junfeng Chen, Prof. Yaoyao He, Tingting Yang, Ying Chen, Wenjing Hong, Yao Zhao, Lukas, Guiying Li, Dr. Yunwen Lei, Dr. Peng Yang, Dr. Bo Yuan, Farhad for either their discussions and suggestions on the research or their help and encouragement in daily life. Thank Chengbin Hou very much for the help before my PhD viva.

Finally, I wish to thank my great parents, my parents in law and my husband for their consistent support, care, encouragement and love. This thesis is dedicated to them.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

List of Abbreviations

AM	Assortative mating
EA(s)	Evolutionary algorithm(s)
ATC	Analytical target cascading
ATC-MO	ATC with multi-objective formulation
CooC	Cooperative co-evolution
ComC	Competitive co-evolution
CCDM	Co-evolutionary concurrent design method
CCDO	Competitive co-evolution for dynamic optimisation
CE	Concurrent engineering
CFS	Communication frequency self-adaptation
COSMOS	Collaborative optimisation strategy for multi-objective system
COSSOS	Collaborative optimisation strategy for single-objective system
DC	Deterministic crowding
DCOP(s)	Dynamic constrained optimisation problem(s)
DO	Dynamic optimisation
DOP(s)	Dynamic optimisation problems
HyperM	Hyper-mutation
LS	Local search
MDO	Multidisciplinary design optimisation
MORDACE	Multidisciplinary optimisation and robust design approaches applied to concurrent engineering
NDCC	Novelty-driven cooperative co-evolution
NDCC-MO	NDCC with multi-objective method
NDCC-MO-a	NDCC-MO with archive
NDCC-SS	NDCC with stochastic selection
NDCC-WS	NDCC with weighted sum method
NDCC-WS-a	NDCC-WS with archive
RI	Random immigrants
SELS	Speciated evolution with local search

CHAPTER 1

INTRODUCTION

1.1 Background

Optimisation problems widely exist in the real world. Inspired from the nature, evolutionary algorithms (EAs) are a class of population-based stochastic optimisation methods [41]. Compared to traditional gradient-based optimisation methods, EAs show some specific advantages. They do not require gradient information of problems and thus can be applied to solve non-differentiable, discontinuous, noisy, and multi-objective optimisation problems. Moreover, they have global search ability and can be conducted in parallel. These advantages help EAs successfully attract researchers' and practitioners' attentions. EAs have become one of the most popular optimisation techniques and have achieved a great success on a variety of real-world applications such as engineering design [115], financial forecasting [7], job shop scheduling [51], music composition [56], drug design [65] and so on.

However, with the rapid development of technology, real-world optimisation problems become increasingly complex. On one hand, optimisation problems may have a lot of decision variables or involve multiple disciplines (e.g., aerodynamics, structures in car design). One such situation occurs in the field of concurrent engineering (CE). Modern products and their design processes have become much more complex than before. The wide use of CE further increases the complexity of product design. As an organisation method,

CE advocates considering all parts or all aspects of the product design simultaneously [63, 130]. This leads to various design problems that have many design variables (if the product to design has a complex physical structure) or require different disciplinary expertise to solve. On the other hand, many optimisation problems change over time these days due to the dynamic environments such as vehicle routing [44], load balancing [134], service composition [57], etc.

For the above-mentioned complex problems, basic EAs tend to perform poorly and thus simply applying them can not solve the problems. The reasons behind this are as follows. First, problems with many decision variables usually have very large search space. When dealing with such problems, EAs can not search efficiently due to the curse of dimensionality. Second, for problems involving multiple disciplines, the designer might not have access to the simulation models of each discipline, or the integration of different analysis codes might be too difficult or costly [58, 90]. As a result, EAs can not be directly applied to solve this kind of problems as a whole. Third, in the context of dynamic problems, the optimum of the problem might change as time goes by. This requires EAs to quickly find the new optimum once the problem changes. However, neither restarting the optimisation after a change nor continuing search on the current population without any change can react quickly to changes [93].

The difficulties in using EAs to handle these complex optimisation problems have been drawing growing interests from academia and industry. Decomposition strategies are introduced into traditional EAs and different types of multi-species EAs appear. A species or sub-population denotes a group of similar individuals capable of interbreeding, but not with individuals from a different group, which is also the definition of species in biology [72]. In this thesis, multi-species EAs are used to denote the evolutionary approaches that explicitly or implicitly maintain multiple species (sub-populations) to evolve different sub-components or search different regions of the search space. According to this definition, multi-population EAs [12, 69] and co-evolutionary EAs [52, 104] are all considered as multi-species EAs.

For problems with many decision variables or involving multiple disciplines, co-evolutionary EAs have been widely applied to address them through decomposing the original problem into smaller sub-problems and evolving each sub-problem with a species (sub-population) of individuals [6, 79, 83, 127, 129]. In another context, multi-population EAs [12, 69] and niching EAs [86, 100] have become one major technique of handling dynamic optimisation problems (DOPs) [12, 69, 86, 100]. They can maintain population diversity by explicitly or implicitly using multiple species (sub-populations) in the population to search different regions of the decision space.

The basis of multi-species EAs is a decomposition of problem or search space. Co-evolutionary EAs depend on the decomposition of decision variables, multi-population EAs depend on the search space decomposition. For some problems, an appropriate decomposition may be known beforehand with a priori knowledge [105]. For example, in the engineering design field, the product decomposition method usually decomposes the design problem based on the predefined product structure and the process decomposition method breaks up the design problem according to engineering discipline [64]. However, in most cases, the decomposed sub-problems are usually related to each other. The performance of a candidate solution for one sub-problem might depend on the solution for another sub-problem, or different sub-problems need to coordinate on their shared decision variables. The difficulty that exists in these cases is how to make different sub-problem optimisation collaborate with each other efficiently when using multi-species EAs.

For many other problems, we may have little priori information about the roles or the number of sub-problems [105]. For example, if the task is to locate different feasible regions for a constrained optimisation problem, we probably will not know beforehand the number of feasible regions or how to allocate individuals to search different feasible regions. Moreover, when facing a dynamic optimisation problem, simply using multiple species (sub-populations) to search different feasible regions of the decision space may not be enough, a good balance between exploration and exploitation is also necessary.

The aforementioned issues pose a demand on efficient decomposition strategies and collaboration strategies when applying multi-species EAs to address such complex optimisation problems. Motivated by this, the focus of this thesis is to develop specific and efficient multi-species EAs with good decomposition and collaboration strategies to address different complex optimisation problems. Section 1.2 will describe the scope of optimisation problems considered in this thesis. In Section 1.3, the research questions of this thesis and their motivations will be clearly explained. After this, the contributions of this thesis will be outlined in Section 1.4. Then, in Section 1.5, the thesis organisation is given and the content of each following chapter is summarised. Finally, Section 1.6 will list the published or submitted papers resulting from the thesis.

1.2 Scope of the Thesis

The range of complex optimisation problems is large and diversified. However, only CE problems and dynamic optimisation problems (DOPs) are of particular interest in this thesis. Concretely, two typical types of CE problems are considered in this thesis. The first problem is the design of a product that has several different parts with respect to one discipline, and the other is the quasi-separable multi-disciplinary design optimisation (MDO) problem in which different disciplines share parts of design variables. Quasi-separable MDO problems are frequently encountered in MDO [123] and a general MDO problem can be easily transformed into a quasi-separable MDO problem [82]. Dynamic optimisation problems include unconstrained and constrained problems. In view of their high popularity in real-world applications, dynamic constrained optimisation problems (DCOPs) are taken into consideration in this thesis.

1.3 Research Questions

As mentioned above, the central point of the thesis is to develop efficient multi-species EAs with good decompositon and collaboration strategies to address CE problems and DCOPs. In this section, a clear explanation of the research questions of the thesis around this central point as well as their motivations is given.

1.3.1 Collaboration Strategies among Sub-Problem Optimisation in CE

Design problems in CE usually have many decision variables or require different disciplinary expertise to solve. To enable concurrent design, decomposition-based optimisation strategies are preferred in CE. In the literature, there exist three decomposition methods, i.e., product decomposition, process decomposition and problem decomposition [64]. Product decomposition is based on the predefined product structure, problem decomposition depends on the dependencies among design parameters, and process decomposition breaks up MDO problems according to engineering disciplines.

The divide-and-conquer idea behind CE is essentially similar to that of cooperative co-evolution (CooC) [77]. CooC usually assumes that the design parameters are divided into several disjoint parts and evolves each part with an EA [105]. Thus, CooC can be directly applied to CE problems in which decomposition is based on the predefined product structure and there are no shared design variables between different sub-problems. In the literature, CooC has also been employed in the field of CE to deal with MDO problems [17, 18, 90, 127].

However, some questions arise when using CooC to address CE problems. First, in most cases, the decomposed sub-problems are usually related to each other, and the performance of a candidate solution for one sub-problem depends on the solution for another sub-problem. In such cases, it has been shown that CooC can easily converge to mediocre stable states [49]. To alleviate this situation, researchers have proposed to

use a large number of collaborators for individual evaluation [98, 49] or employ novelty search methods [47, 49] to compare individuals based not only on their fitness but also their novelty. However, these existing methods are computationally inefficient and thus restricting the application of CooC in CE. This will further discussed in Section 2.4.1 in Chapter 2.

Second, CooC assumes that the design parameters are divided into several disjoint parts. However, such a decomposition does not necessarily hold for CE since the decomposition is mainly motivated by the real-world applications, and it is likely that different sub-problems share the same design parameters in MDO problems. Although in the above-mentioned CooC work for MDO problems [17, 18, 90, 127], to make existing CooC methods directly applicable to CE, the shared design parameters are still handled as if they are normal parameters and decomposed into several fully disjoint parts, such a strategy also introduces additional questions like which shared variable should be evolved with which sub-problem, and restricts the efficiency of the whole CE procedure.

Third, as the decomposed sub-problems are usually related to each other, when using CooC to solve CE problems, different sub-populations in CooC should communicate with each other. Frequent communications help to ensure that the latest information from other sub-populations is used in optimisation, but slow down the parallel CooC and thus the computational efficiency. Infrequent communications help to increase parallelism and thus computational efficiency, but can lead to optimisation in a sub-population based on out-of-date information from other sub-populations. This brings a non-trivial issue of how often different sub-populations should communicate with each other, which is also a very important research topic in CE. However, although some communication strategies have been proposed in CE [62, 74, 75, 108, 131], they were mainly developed for concurrent design of dependent tasks rather than interdependent tasks which hold for the two CE problems considered in this thesis. In the field of CooC, there are few studies conducted along this direction especially for the situation in which communication incurs a cost.

For the above reasons, this thesis firstly aims to investigate CooC-based design op-

timisation in the context of CE, and study the question of *how to make sub-problems (sub-populations) collaborate with each other efficiently when using CooC to address CE problems*, which actually includes three sub-questions: *how to better evaluate individuals in each sub-population when using CooC to address the first type of CE problem* (**sub-question 1**), *how to better handle the shared design variables when using CooC to address the second type of CE problem* (**sub-question 2**), and *how often different sub-populations should communicate with each other in the CooC-based concurrent design optimisation* (**sub-question 3**).

To answer these questions, in Chapter 3, a computationally efficient novelty-driven CooC method is proposed to address the first type of CE problem. Moreover, a novel CooC-based concurrent design method is given to address the second type of CE problem. Chapter 3 also studies how the communication frequency among sub-populations affects the performance of these two CooC-based methods, and introduces a self-adaptive method to adapt the communication frequency during the optimisation process.

1.3.2 Niching Methods for Dynamic Constrained Optimisation

In the field of concurrent engineering, changes in the design objectives or constraints usually happen suddenly due to the varying customer needs, and fast response to the changes is required [110, 130]. This arises the appearance of dynamic constrained optimisation problems (DCOPs) in CE. Besides, DCOPs also widely exist in other fields due to the dynamic environments, such as vehicle routing [44], load balancing [134], service composition [57], etc. In a DCOP, either the objective function or the constraints, or both, may change over time due to dynamic environments. This causes the optimum of the problem to change as time goes by. Therefore, DCOPs require an optimisation algorithm to quickly find the new optimum once the problem changes [93].

Addressing DCOPs needs to combine dynamic optimisation strategies and constraint handling techniques. However, directly combining them may not perform effectively. In [94], the authors combined two popular dynamic strategies, random-immigrants (RI)

[50] and hyper-mutation (HyperM) [22], separately with a constraint handling method, the penalty function [87], to address DCOPs. However, it has been found that both RI and HyperM are not so effective in solving DCOPs as in solving unconstrained or bound-constrained dynamic optimisation problems. The reasons are as follows. Although RI/HyperM generates and introduces some random solutions into the current population after a change is detected, most of them are infeasible and thus being rejected by the used penalty function. As a result, they can not perform effectively when facing DCOPs in which the global optimum moves from one feasible region to another disconnected one as this kind of DCOPs need an infeasible path connecting two disconnected feasible regions.

Therefore, the autors in [94] suggested using constraint handling techniques that can accept diversified infeasible solutions or tracking moving feasible regions when dealing with DCOPs. In the literature, researchers have carried out some work to allow diversified infeasible solutions distributed in the whole search space. For example, the repair method [112] was applied to handle constraints in DCOPs in [5, 92, 97]. In [15, 16], constraint handling techniques that accept both feasible and infeasible solutions are employed. However, repair methods usually need a lot of feasibility checkings. The use of constraint handling techniques that accept both feasible solutions and infeasible solutions might not react quickly when the global optimum moves to another disconnected feasible region especially when the feasible region in which the new global optimal solution is located is far away.

Intuitively, if the optimisation algorithm already has individuals in a feasible region, it will react quickly when the global optimum moves to this feasible region. One method to implement this is to locate multiple feasible regions. The location of multiple feasible regions can also maintain the diversity of population. This can help to react quickly in other types of dynamic changes. However, little work has been done along this direction before the year of 2016.

To locate multiple feasible regions, individuals in the population need to be assigned to search different parts of the search space. This brings an issue of how to divide the

search space and allocate individuals to different feasible regions. As the feasible regions in DCOPs might move due to environmental changes, an optimisation algorithm for DCOPs should also be able to track moving feasible regions. Although these issues have been rarely studied in DCOPs, how to locate multiple optima has been studied a lot in the field of multi-modal optimisation [70, 118, 128]. One of the major existing techniques is the use of niching methods [113] which explicitly or implicitly maintain multiple sub-populations to search different promising regions of the decision space. In the field of unconstrained DOPs or DCOPs with bounded constraints, niching EAs have also been applied to locate multiple optima and track multiple moving optima [12, 69, 86, 100].

Motivated from these, this thesis secondly aims to apply niching methods to address DCOPs and study the question of *whether niching method can help better solve DCOPs*. Research questions of *how to locate multiple feasible regions in a DCOP* and *how to track moving feasible regions in a DCOP* are studied. As niching methods focus on exploration, newly changed optima might not be found quickly as promising regions are not exploited sufficiently. Therefore, the question of *how to make a good trade-off between exploration and exploitation* is also investigated.

To close research gap and answer these questions, Chapter 4 introduces a speciation-based EA to address the challenges of DCOPs. It employs deterministic crowding and assortative mating to allocate individuals to search different feasible regions, and applies a local search strategy to promote exploitation of the promising regions. To track moving feasible regions, it uses the simple and parameter-free feasibility rules [85] which prefer feasible solutions to infeasible ones to deal with constraints, and adds random solutions into the population to introduce diversity once a change is detected.

1.3.3 Competitive Co-evolution for Fast-Changing Dynamic Constrained Optimisation

Dynamic optimisation problems require an optimisation algorithm to quickly find the new optimum once the problem changes. Thus, a good dynamic optimisation algorithm should

satisfy either of the following two conditions:

1. It is able to track the moving optimum once the problem changes.

2. It can obtain good initial individuals once the problem changes.

Existing dynamic optimisation approaches can be categorised according to the two conditions. Diversity-driven approaches (e.g., diversity introducing or maintaining methods and multi-population methods [93]) satisfy the first condition. Prediction approaches and memory approaches satisfy the second condition.

In Section 1.3.2, the idea of addressing DCOPs by locating and tracking feasible solutions aims to satisfy the first condition. However, doing this can not have enough time to locate good solutions when the problems change rapidly. Fast-changing DCOPs need optimisation algorithms that satisfy the second condition. In the literature, only prediction approaches [37, 38] and memory approaches [107] that have been developed for DCOPs satisfy the second condition. But, they are inappropriate when a DCOP is not cyclic or predictable. Moreover, as the time for optimisation is generally very short, the memory approaches can not have optimal solutions to archive and thus the prediction approaches can not have useful samples for accurate prediction even when the DCOP to solve is cyclic or predictable. These motivate the thesis to further address DCOPs with fast changes from the aspect of satisfying the second condition.

Intuitively, suppose a set of good solutions is already obtained and stored in the system before it goes online, and for any possible change induced by the environments, one of the solutions can be efficiently modified to get a good solution. Then, the second condition would be satisfied, and the DOP or DCOP in the online phase would not be as challenging as we expected. Such an idea of preparing beforehand is just like what is done in reality. In most real-world scenarios, one may have pretty long time to improve a system (and the algorithm behind it) before putting it into full use. For example, a company may spend months to polish its backend system before launching a new service composition system, and once the system is put online, it may need to react to changes on a daily

basis. Considering these, this thesis thirdly studies the questions of *whether identifying a set of promising solutions offline could be beneficial to online solve DCOPs* and *how to achieve such a solution set*.

To search a set of promising solutions for a DCOP beforehand, both the solution space and the environment space need to be considered. One naive method is to find the optimal solution under each environment. However, this process is time intractable. An alternative method is to sample the environments. To have a solution set of good coverage on all environments, the sampled environment each time should be the one that challenges the current solution set most. Moreover, the solution set needs to be updated to conquer the newly sampled environments. The relationship between the solution set and environments in this way is similar to the relationship between the host population and the parasite population in competitive co-evolution (ComC) [52, 109]. Inspired from these, this thesis employs ComC to search a set of promising solutions for DCOPs, and studies the question of *whether decomposing DCOPs into solution population and environment population and co-evolving them using ComC can find a better solution set*.

To answer the aforementioned questions, Chapter 5 introduces a new dynamic handling strategy, competitive coevolution for DCOPs, which searches for a set of good solutions using ComC in an offline manner, and conducts online optimisation by using this set for intialisation each time a change is detected. This new method is tested on a DCOP benchmark and compared to existing methods. Furthermore, ComC is compared to a random sampling method in searching the solution set.

1.4 Contributions of the Thesis

This thesis tries to close research gap and answer the research questions listed in Sections 1.3.1 to 1.3.3. The contributions of the thesis are summarised as follows. They will be discussed in more details in Chapter 6.

- A new novelty-driven cooperative co-evolution (CooC) algorithm, which uses a com-

putationally efficient novelty calculation and applies a stochastic selection strategy to decide whether evaluating individuals based on their fitness or novelty. The proposed algorithm is designed for the first type of CE problems as defined in Eq.(2.1). In addition to its computational efficiency, it is shown to obtain designs of higher quality on the single universal electric motor (UEM) design problem. This is an answer to the **sub-question 1** in Section 1.3.1, and can be found in Chapter 3.

- A novel CooC-based concurrent design method for quasi-separable MDO problems, which enables concurrent design by using consistency constraints to deal with common variables among different disciplines and stochastic ranking method [111] to handle the constraints. It achieves designs of higher quality in comparison to other MDO methods on a general MDO problem and the design of multiple UEMs that have common design variables. This is an answer to the **sub-question 2** in Section 1.3.1, and can be found in Chapter 3.

- A systematic study on how the communication frequency among sub-problem optimisation affects the final design in CE. The optimal communication frequencies under different communication costs are reported for the two above-mentioned CooC-based concurrent design methods. An effective self-adaptive method is then given for them to adapt the communication frequency during optimisation. This is an answer to the **sub-question 3** in Section 1.3.1, and can be found in Chapter 3.

- A new approach to deal with environmental changes in DCOPs, which uses speciation methods to locate and track moving feasible regions. This approach can react faster when the global optimum moves from one feasible region to another disconnected feasible region in comparison to other approaches. This is an answer to questions in Section 1.3.2, and can be found in Chapter 4.

- A novel strategy to deal with environmental changes in fast-changing DCOPs, which uses competitive co-evolution (ComC) to find a set of good solutions beforehand and use it for initialisation once an environmental change happens. It is shown to react

more quickly to environmental changes especially for DCOPs with fast changes through empirical studies. This is an answer to questions in Section 1.3.3, and can be found in Chapter 5.

- A wider application of multi-species EAs. As far as we know, for the first time, this thesis investigates the use of speciation methods in the context of DCOPs, and attempts to employ ComC to address fast-changing DCOPs. This thesis widens the application scope of multi-species EAs on complex optimisation problems.

1.5 Organisation of the Thesis

The remaining content of the thesis is organised as follows. Chapter 2 reviews the basic information of multi-species EAs and existing research work related to this thesis. The purpose of this chapter is to present a general review of what has been done in terms of solving CE problems and DCOPs using EAs and multi-species EAs, and demonstrate the challenges that still exist.

Chapter 3 introduces two new concurrent design methods based on the parallel CooC framework to address two kinds of CE problems. Additionally, Chapter 3 studies how the communication frequency among sub-populations affects the performance of the proposed CooC methods, and introduces a self-adaptive method to adapt the communication frequency during optimisation. Experiments conducted on universal electric motor (UEM) design problems and a geometric programming problem are detailed in this chapter.

Chapter 4 details a new speciation-based method, SELS, to address DCOPs. The SELS method is tested on DCOPs benchmark problems and compared to several state-of-the-art methods. Experimental results are shown in this chapter.

Chapter 5 presents a new dynamic optimisation approach, CCDO, to deal with DCOPs from a different aspect. The CCDO is tested on DCOPs benchmark test problems and experimental studies are presented to show its efficiency.

Finally, in Chapter 6, contributions of the thesis are summarised and future research

directions are suggested.

1.6 Publications Resulting from the Thesis

The published or submitted papers resulting from the thesis are listed as follows.

1. Referred or Submitted Journal Papers

[1]. X. Lu, K. Tang, B. Sendhoff and X. Yao, "A review of concurrent optimisation methods", *Int. J. of Bio-Inspired Computation*, Vol.6, No.1, pp.22 - 31, 2014.

[2]. X. Lu, S. Menzel, K. Tang and X. Yao, "Cooperative co-evolution based design optimisation: a concurrent engineering perspective", *IEEE Transactions on Evolutionary Computation*, Vol. 22, Issue 2, pp. 173-188, 2018.

[3]. X. Lu, K. Tang and X. Yao, "Competitive co-evolution for dynamic constrained optimisation", *submitted to IEEE Transactions on Evolutionary Computation.*

2. Referred Conference Papers

[4]. X. Lu, S. Menzel, K. Tang and X. Yao, "The performance effects of interaction frequency in parallel cooperative coevolution", In: Dick G. et al. (eds) *Simulated Evolution and Learning. SEAL 2014. Lecture Notes in Computer Science*, vol 8886. Springer, Cham.

[5]. X. Lu, K. Tang, X. Yao, "Speciated evolutionary algorithm for dynamic constrained optimisation", In: Handl J., Hart E., Lewis P., Lpez-Ibez M., Ochoa G., Paechter B. (eds) *Parallel Problem Solving from Nature PPSN XIV. PPSN 2016. Lecture Notes in Computer Science*, vol 9921. Springer, Cham.

The following lists the number of the published or submitted papers that are (partly) presented in each chapter of the thesis:

- Chapter 2: publication [1]
- Chapter 3: publication [2,4]
- Chapter 4: publication [5]
- Chapter 5: publication [3]

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

This chapter introduces the background knowledge and reviews existing research work related to this thesis. Section 2.1 gives the problem formulations for the two typical types of concurrent engineering (CE) problems and dynamic constrained optimisation problems (DCOPs) that are considered in this thesis. In Section 2.2, an introduction to evolutionary algorithms (EAs) is given. Section 2.3 reviews different types of EA extensions which include niching EAs, multi-population EAs, cooperative co-evolution (CooC) and competitive co-evolution (ComC). In Section 2.4, the issues of individual evaluation and communication frequency in CooC as well as existing methods are introduced and discussed. In Section 2.5, a review of using EAs to address multi-disciplinary design optimisation (MDO) problems is presented, and Section 2.6 gives an overview of existing dynamic constrained optimisation methods. Finally, Section 2.7 summarises this chapter.

2.1 Problem Formulations

2.1.1 Two Types of CE Problems

In this thesis, two types of complex design problems, which are frequently encountered in CE, are considered. They are:

1. Mono-disciplinary product design with multiple parts,

2. Quasi-separable MDO problems.

For the first type of design problems, assuming the product is composed of m parts and f denotes the performance metric of the design, the problem formulation can be given as follows:

$$\min_{\boldsymbol{x}_1,\boldsymbol{x}_2,...,\boldsymbol{x}_m} f\left(\boldsymbol{x}_1,\boldsymbol{x}_2,...,\boldsymbol{x}_m\right) \tag{2.1}$$

where $\boldsymbol{x}_i$ denotes the design parameter vector related to the i-th part of the product $(i = 1, 2, ..., m)$. Note that we assume throughout this thesis that the decomposition of the product is known beforehand and there are no shared design variables between different subproblems.

For the second type of design problems, different disciplines aim to find a complete design that improves different physical aspects of a product. A quasi-separable MDO problem [123] with m disciplines can be formulated as:

$$\min_{\boldsymbol{x}_1,\boldsymbol{x}_2,...,\boldsymbol{x}_m,\boldsymbol{z}} \sum_{i=1}^{m} f_i(\boldsymbol{x}_i,\boldsymbol{z}) \tag{2.2}$$

where $(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_m, \boldsymbol{z})$ denotes the design parameter vector of the entire design. $\boldsymbol{z}$ is the shared design parameter vector among disciplines. $\boldsymbol{x}_i$ denotes the local design parameter vector with respect to the i-th discipline and f_i denotes the objective of the i-th discipline $(i = 1, 2, ..., m)$. The final design from each discipline must be consistent on $\boldsymbol{z}$. Note that we assume throughout this thesis that the shared design variables are known beforehand for this kind of problem.

2.1.2 Dynamic Constrained Optimisation Problems (DCOPs)

Without loss of generality, the dynamic constrained optimisation problems considered in this thesis have the following formulation:

$$\begin{aligned} &\min_{\boldsymbol{x}} \ f(\boldsymbol{x}, \boldsymbol{\alpha}(t)) \\ &\text{subject to} : g_i(\boldsymbol{x}, \boldsymbol{\alpha}(t)) \leq 0, i = 1, 2, ..., k \end{aligned} \tag{2.3}$$

where $\boldsymbol{x}$ denotes the decision variable vector, k denotes the number of constraints, $\boldsymbol{\alpha}(t)$ is the vector of environmental parameters. They vary at a certain frequency as time goes by. This kind of dynamic problems require an optimisation method to quickly find the new optimal solution each time it changes.

2.2 Introduction to EAs

Evolutionary algorithms (EAs), originated in 1960s, are a class of stochastic optimisation methods inspired from natural evolution [40, 41, 36]. They maintain a population of individuals, each of which denotes a candidate solution to the optimisation problem, and improve the population through selecting the fittest ones and generating offsprings generation by generation. For single-objective optimisation problems, the fitness of an individual is usually set to the objective function value of the individual (if it is a maximising problem) or the minus of the objective function value (if it is a minimising problem).

Without loss of generality, assume a single-objective optimisation problem has the following formulation:

$$\min_{\boldsymbol{x}} \ f(\boldsymbol{x}) \tag{2.4}$$

where $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ denotes the vector of decision variables and n is the number of decision variables. The range for each x_i is L_i and U_i, i.e. $x_i \in [L_i, U_i]$. $f : \Omega \subseteq \Re^n \rightarrow \Re$ is the objective function of the problem. Note that a maximising optimisation problem can be easily tranformed to a minimising optimisation problem by multiplying the objective

function f with -1.

The process of using an EA to address this opimisation problem can be found in Algorithm 1. The EA begins with a population of randomly generated candidate solutions, $\boldsymbol{P}_G = \{\boldsymbol{x}_{i,G} | i = 1, 2, ..., NP\} (G = 0)$. Here, G denotes the generation number, NP is the population size, and each $\boldsymbol{x}_{i,G}$ denotes a candidate solution to the optimisation problem in Eq. (2.4). After initialisation, the EA iteratively uses selection, recombination, and mutation operations at each generation G to evolve the population until a stopping criterion is met. The best individual $\boldsymbol{x}_{i,G}$ in the final population is the output.

Algorithm 1 The Framework of EA

1: Initialise a population $\boldsymbol{P}_G = \{\boldsymbol{x}_{i,G} | i = 1, 2, ..., NP\}$
2: Evaluate $\boldsymbol{P}_G$ with the objective function f
3: **while** the stopping criterion is not met **do**
4: Select parents $\boldsymbol{P}_{\text{par}}$ from $\boldsymbol{P}_G$ based on the fitness of each $\boldsymbol{x}_{i,G}$
5: Conduct recombination operation on $\boldsymbol{P}_{\text{par}}$ to get $\boldsymbol{P}_{\text{off}}$
6: Conduct mutation operation on $\boldsymbol{P}_{\text{off}}$ to get $\boldsymbol{P}'_{\text{off}}$
7: Evaluate $\boldsymbol{P}'_{\text{off}}$ with the objective function f
8: Select NP individuals $\boldsymbol{P}_{\text{S}}$ from $\boldsymbol{P}_G \cup \boldsymbol{P}'_{\text{off}}$ according to individuals' fitness
9: Set $\boldsymbol{P}_{G+1} = \boldsymbol{P}_{\text{S}}$
10: Set $G = G + 1$
11: **end while**

Note that most EAs follow a similar process as in Algorithm 1. Different EAs differ from each other in the use of different recombination, mutation or selection operators. The most popular EAs include genetic algorithm (GA) [55, 29], evolutionary strategy (ES) [10, 8], evolutionary programming (EP) [42], genetic programming (GP) [61], ant colony optimisation (ACO) [23], particle swarm optimisation (PSO) [35, 102], differential evolution (DE) [120, 121], and so on. Each of them has been studied a lot and has various algorithm variants nowadays.

Compared to traditional optimisation methods, e.g. gradient-based methods, hill climbing, and simulated annealing, EAs show some specific advantages. First, they search with a population of individuals with the aim of searching globally. Second, they use the fitness of individuals to guide the search direction, and thus can be applied to solve non-differentiable, discontinuous, or black-box optimisation problems. Third, the

individual-based evolutionary operation and evaluation enable the parallel execution of an EA. Therefore, EAs have become one of the most popular optimisation techniques and been applied to various real-world applications, e.g. engineering design [115], financial forecasting [7], job shop scheduling [51], music composition [56] and drug design [65].

2.3 Extensions to Traditional EA Model

However, with the rapid development of society, real-world optimisation problems become increasingly complex. As a result, traditional EA models are not adequate to deal with them. For example, when an optimisation problem involves many decision variables, optimising all the decision variables as a whole might not be the best approach due to the curse of dimensionality. Also, many engineering optimisation problems require the location of multiple optima in the search space but traditional EA models tend to converge to only one solution due to the genetic drift from the selection operator. To better address these problems, various extensions to the basic EAs have been proposed. In the following parts, different types of EA extensions will be introduced.

2.3.1 Niching Techniques and Multi-Population EAs

A niche in the optimisation is commonly referred to an area of the fitness landscape where only one peak resides [72]. Niching methods are developed to reduce the effect of genetic drift in the traditional EAs by altering the selection operator to provide selection pressure within, but not across regions of the search space [118]. Through doing this, niching methods can maintain individuals in different regions and thus permit EAs to locate multiple optima in parallel. In the literature, there exist various means of niching implementation.

One classic niching method is fitness sharing [46] which makes individuals share their fitness with individuals nearby and use the shared fitness in replace of their original fitness

for selection. The shared fitness for an individual in fitness sharing is defined as:

$$f_{shared}(i) = \frac{f_{original}(i)}{n_i} \quad \text{where} \quad n_i = \sum_{j=1}^{N} Sh(d_{ij}) \tag{2.5}$$

where $f_{original}(i)$ represents the original fitness of the i-th individual, and $Sh(d_{ij})$ denotes the sharing function:

$$Sh(d_{ij}) = \begin{cases} 1 - (\frac{d_{ij}}{\delta_{share}}) & \text{if } d < \delta_{share}; \\ 0 & \text{oterwise} \end{cases} \tag{2.6}$$

Here, α and δ_{share} denote the scaling factor and the niche radius, respectively. Through the use of fitness sharing, the growth of the number of individuals near one peak is restricted and individuals are encoraged to cluster around different peaks (niches).

Another classic method is crowding, originally proposed by De Jong in [29], which does replacement operation between similar individuals. In crowding, for each newly generated offspring, a sample of individuals are taken from the current population and the most similar individual to the offspring is replaced. To reduce the replacement errors and eliminate the setting of the sampling number, Mahfoud further improved crowding and proposed the deterministic crowding (DC) in [80]. Algorithm 2 gives the pseudo-code of DC.

Algorithm 2 Deterministic Crowding [80]

```
Randomly pair all individuals in the population
for each pair of individuals, p1 and p2, do
    Generate two offspring, o1 and o2, based on EA operators
    if dist(p1, o1) + dist(p2, o2) ≤ dist(p1, o2) + dist(p2, o1) then
        p1 = fitter(p1, o1)
        p2 = fitter(p2, o2)
    else
        p1 = fitter(p1, o2)
        p2 = fitter(p2, o1)
    end if
end for
```

The DC method pairs all population elements randomly and generates two offspring for each pair based on EA operators. Selection is then operated on these four individuals,

and a similarity measure is used to decide which offspring competes against which parent. The offspring will replace the compared parent and enter next generation if it is fitter. Mengshoel et al. further improved DC by employing a probabilistic replacement strategy in [84]. Suppose individual u and v are competing against each other, in the probabilistic replacement, the probability of u winning and replacing v is set to:

$$p_u = \frac{f_u}{f_u + f_v} \tag{2.7}$$

where f_u and f_v are the fitness of u and v, respectively.

The authors in [70] proposed a species conservation method which partitions the population into a set of dominated species and marks the best individual in each species as species seed. At the beginning, this method sorts the population from the best to the worst. Then, for each individual that does not belong to any species, the similarity distance between this individual and the species seed in each existing species is calculated. If the distance is less than the pre-defined radius, this individual will belong to the species. Otherwise, this individual will be considered as a new species. In later generations, to conserve species, each species seed will be compared with the worst offspring individual whose distance to the species seed is less than the species radius and not replaced by any species seed before, and then the species seed will replace it if the species seed is better. If there does not exist such an offspring individual, the species seed will replace the worst offspring individual that is not replaced by any species seed before. In [71], the authors further combined this species partition method with PSO. In this species-based PSO method, the population is partitioned into species at every generation. Each species seed is used as the local best for individuals in the species and replaces the global best in PSO to generate offspring individuals. This species-based PSO method has been later applied to address dynamic optimisation problems [100].

Other niching techniques include clearing method, clustering method, restricted tournament selection, and so on. More information about them can be found in [72, 118].

Note that niching EAs are actually firstly devised to maintain the population diversity of EAs and later extended to deal with multi-modal optimisation problems [72]. Furthermore, niching EAs are nowadays extended to deal with dynamic optimisation problems as they can maintain population diversity and permit tracking moving optima. A recent survey paper in this direction can be found in [69].

2.3.2 Cooperative Co-evolution (CooC)

The performance of EAs deteriorates as the dimension of the problem increases [95]. To make EAs have better scalability to high-dimensional problems, Potter and De Jong proposed cooperative co-evolution (CooC). As a biological concept, co-evolution refers to that two or more species evolve simultaneously and affect each other's evolution with the coupled fitness. The interaction of different species in co-evolution can be cooperative (e.g. mutualism) or competitive (e.g. predator and prey, host and parasite). Algorithm 3 shows the framework for CooC.

CooC was inspired from the cooperative interaction between different species. In CooC, the decision variables are firstly decomposed into several sub-components. Then, these sub-components will be evolved in different sub-populations, each with a specified EA. Fitness evaluation for each sub-population individual is carried out by combining it with representative individuals (usually the current best individuals) from the other sub-populations. The whole optimisation process of CooC is decomposed into several cycles. In every cycle, each sub-population is evolved with a specified EA for a fixed number of generations. At the end of each cycle, different sub-populations communicate with each other the best individuals in their current populations, which are used as representative individuals to evaluate the individuals in the other sub-populations in the next cycle.

2.3.3 Competitive Co-evolution (ComC)

Competitive co-evolution was proposed to model the competitive interaction between different species in biology (e.g. predator and prey, host and parasite) [109]. In [52], ComC was first applied to search a good set of test cases for sorting networks. In the software engineering field, finding a good test suite is of the same importance as developing the software. The work in [52] treated the sorting networks as host population and test suite as parasite population. The fitness of each sorting network in the host population is set to how many test samples in the parasite population it correctly sorts, and the fitness of each test sample in the parasite population is set to how many sorting networks in the host population that can not correctly sort this test sample. Through doing this, the sorting networks and the test suite are then evolved simultaneously.

Algorithm 3 The Framework of CooC

Initialise $\boldsymbol{P}_{1,0}, \boldsymbol{P}_{2,0}, ..., \boldsymbol{P}_{m,0}$ for m sub-components
Randomly combine $\boldsymbol{P}_{1,0}, \boldsymbol{P}_{2,0}, ..., \boldsymbol{P}_{m,0}$ into a big population $\boldsymbol{P}$
Evaluate the population $\boldsymbol{P}$ with the objective function
Select the best individual in $\boldsymbol{P}$ denoted as: $\boldsymbol{C} = (\boldsymbol{x}_1^{best}, \boldsymbol{x}_2^{best}, ..., \boldsymbol{x}_m^{best})$
Set $cycle = 0$
while $cycle <$the maximum number of cycles **do**
  for each $\boldsymbol{P}_{i,cycle}$ **do**
    Set $G = 0$ and $\boldsymbol{P}_{i,G} = \boldsymbol{P}_{i,cycle}$
    while $G <$the maximum number of generations **do**
      Conduct evolutionary operation on $\boldsymbol{P}_G$ to generate $\boldsymbol{P}_{G+1}$
      for each $\boldsymbol{x}_{i,G+1}^j$ in $\boldsymbol{P}_{i,G+1}$ **do**
        Set $\boldsymbol{C}_i = (\boldsymbol{x}_1^{best}, \boldsymbol{x}_2^{best}, ..., \boldsymbol{x}_{i,G+1}^j, \boldsymbol{x}_{i+1}^{best} ..., \boldsymbol{x}_m^{best})$
        Evaluate $\boldsymbol{P}_{\mathrm{G+1}}$ with the objective function f
      end for
      Set $G = G + 1$
    end while
    Set $\boldsymbol{x}_i^{best}$ as the best individual in $\boldsymbol{P}_{i,G}$
    Set $\boldsymbol{P}_{i,cycle} = \boldsymbol{P}_{i,G}$
  end for
  Set $\boldsymbol{C} = \{\boldsymbol{x}_1^{best}, \boldsymbol{x}_2^{best}, ..., \boldsymbol{x}_m^{best}\}$
  Set $cycle = cycle + 1$
end while

Algorithm 4 gives the competitive co-evolutionary framework for two populations, host population $\boldsymbol{P}_h$ and parasite population $\boldsymbol{P}_p$. In Algorithm 4, the two populations are

evolved with specific EAs. f denotes the fitness function. In this algorithm, the fitness of an individual in one population is evaluated with how many individuals in the other population it can defeat. Through such a competition and evolution pattern, both the host population and parasite population will be enhanced.

Algorithm 4 The Framework of ComC

```
1: Initialise host population P_{h,G} and parasite population P_{p,G}
2: Set G = 0
3: while G <the maximum number of generations do
4:    for each x_i^h in P_{h,G} do
5:       Set count = 0
6:       for each x_i^p in P_{p,G} do
7:          if x_i^h beats x_i^p then
8:             count = count + 1
9:          end if
10:      end for
11:      Set f(x_i^h) = count
12:   end for
13:   Select parent population P_{h,par} from P_{h,G}
14:   Do evolutionary operation on P_{h,G} to get P_{h,G+1}
15:   for each x_i^p in P_{p,G} do
16:      Set count = 0
17:      for each x_i^h in P_{h,G} do
18:         if x_i^p beats x_i^h then
19:            count = count + 1
20:         end if
21:      end for
22:      f(x_i^p) = count
23:   end for
24:   Select parent population P_{p,par} from P_{p,G}
25:   Do evolutionary operation on P_{p,G} to get P_{p,G+1}
26:   Set G = G + 1
27: end while
```

The fitness of individuals in ComC is subjective. Therefore, it is suitable for problems without known fitness function or the fitness of an individual is very hard to compute. In addition to sorting networks, the idea of CooC has been extensively applied to solve real-world problems such as iterated prisoner's dilemma problem [20, 27] and playing chess games [19]. However, one requirement that restricts the application range of ComC is that the problem needs to be hand-decomposed into two antagonistic sub-components

before using ComC [105].

2.4 Issues in Cooperative Co-evolution

2.4.1 Individual Evaluation

Cooperative co-evolution encounters several issues as the fitness of individuals in each sub-population depends highly on the representative individuals (also noted as collaborative individuals) exchanged from the other sub-populations. It can easily be misled by the chosen collaborators and get trapped in the suboptimal equilibrium states in which changing each of the team members will result in lower performance [49]. Moreover, CooC tends to identify local optima that have large basins of attraction [14]. However, these local optima may not correspond to global optima. This is known as *relative overgeneralisation* [99].

To prevent CooC from premature convergence to mediocre stable states, it is found that each individual in CooC may need to be evaluated with a large number of collaborators [98, 49]. However, this will increase the cost in evaluating each individual and thus is inefficient especially when the CooC involves more than 3 sub-populations. To address this issue, researchers in [49, 47] have proposed considering not only the fitness but also the novelty of individuals when doing selection on individuals in each sub-population.

Novelty search is a recently proposed evolutionary approach to solve deceptive problems [68]. In the original novelty search method, individuals are scored based on their behavioural novelty rather than fitness. This scheme makes the evolution continuously explore the regions of individuals with behavioural innovation rather than converging to one region. A novelty metric was proposed in [66] to measure how far an individual is from other individuals in the behaviour space, which calculates the novelty score of each individual as follows:

$$nov(\boldsymbol{x}) = \frac{1}{k}\sum_{i=1}^{k} dist(\boldsymbol{x}, \boldsymbol{\mu}_i) \tag{2.8}$$

where $\boldsymbol{\mu}_i$ is the i-th nearest neighbour of the individual $\boldsymbol{x}$ according to the distance

metric *dist*. The neighbours include the other individuals in the current population and optionally archived past individuals. In previous studies, the archive is composed of the most novel individuals [66, 73] or stochastically selected individuals from every generation [67]. This kind of novelty-based optimisation has shown better performance than the fitness-based optimisation methods in many different applications.

The use of novelty as the sole criterion does not always result in satisfying outcomes. It has been found that selecting individuals based on only novelty scores does not necessarily lead to a high average fitness [26]. In the literature, researchers have combined novelty and fitness objectives together to score individuals and shown through experimental studies that this is a more effective way to make use of novelty [49]. Different combination methods have been proposed in the literature [26, 48, 88]. In [26], the authors achieved the combination through a linearly weighted sum of them as shown below:

$$score(\boldsymbol{x}) = (1-\rho) * f_{\text{norm}}(\boldsymbol{x}) + \rho * nov_{\text{norm}}(\boldsymbol{x}) \tag{2.9}$$

where ρ is used to control the importance of the fitness and novelty and kept fixed during the optimisation process, and $f_{\text{norm}}(\boldsymbol{x})$ and $nov_{\text{norm}}(\boldsymbol{x})$ denote the normalised fitness and novelty, respectively. The normalisation process is proceeded according to:

$$f_{\text{norm}}(\boldsymbol{x}) = \frac{f(\boldsymbol{x}) - f_{\text{min}}}{f_{\text{max}} - f_{\text{min}}}, nov_{\text{norm}}(\boldsymbol{x}) = \frac{nov(\boldsymbol{x}) - nov_{\text{min}}}{nov_{\text{max}} - nov_{\text{min}}} \tag{2.10}$$

where f_{min} and f_{max} are the lowest and highest fitness in the current population, respectively; nov_{min} and nov_{max} are the corresponding lowest and highest novelty scores. In [88], a multi-objective evolutionary algorithm, NSGA-II [31], was employed to balance between the novelty and fitness objectives. The weighted sum method and the multi-objective combination method were applied in CooC in [47] and [49], respectively.

However, these novelty-driven CooC methods have some disadvantages. First, the time complexity of calculating the novelty for a whole population according to Eq. (2.8) without considering archived individuals is $\mathcal{O}(NP^2 * \text{complexity}(dist) + NP^2 log(NP))$

(NP denotes population size), which is not computionally efficient. Second, using either a fixed weight or multi-objective method might not be the best trade-off approach between exploration and exploitation as it might be better to change the emphasis to exploration or exploitation as the evolution proceeds.

2.4.2 Communication Frequency in CooC

Another issue in CooC is about the setting of interaction frequency among different sub-populations. The interaction frequency denotes the number of maximum evolution generations before different sub-populations communicate with each other the collaborative individuals. In the literature, one systematic study to investigate the performance effects of interaction frequency was conducted by Popovic and De Jong in [103]. In this paper, the sequential update scenario of CooC was studied in which sub-populations take turns in evolution. That is, during each cycle, only one sub-population is active and the others are frozen. At the end of each cycle, the sub-population that was evolved communicates its best individual to the other frozen sub-populations, and then they switch roles.

By using different cycle sizes and dynamics analysis of best individuals, the work in [103] showed that the performance effect of interaction frequency is dependent on the problem property called best-response curves. It also gave some knowledge about how the performance changes with the increase of cycle size on different kinds of best-response curves. However, as this work considered the sequential scenario of CooC, the communication cost between different sub-populations is not considered. The knowledge learned in this study can not be applied to the use of parallel CooC in which different sub-populations are evolved simultaneously.

Considering this, we have conducted a study on the performance effect of communication frequency in a parallel CooC in [76]. It is found that the best communication frequency changes when the problem and communication cost change. Moreover, the best communication frequency of the parallel CooC framework might change during the whole optimisation process. Therefore, it is not a trivial work to find a good communication

frequency for a new problem. How to set the communication frequency in parallel CooC is still worth studying.

2.5 Use of Evolutionary Algorithms in MDO

A general MDO problem involves multiple disciplines that share some design variables and are coupled by coupling variables. Fig. 2.1 shows a two-disciplinary system. In this figure, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ denotes local variable vectors for disciplines 1 and 2, respectively; $\boldsymbol{z}$ stands for the common variable vector; f_1 and f_2 denote the corresponding objective functions of disciplines 1 and 2; g_1 and g_2 stand for the constraints of disciplines 1 and 2, respectively; f_{system} and g_{system} are the system objective and constraint functions, respectively. The f_{system} is a function of f_1 and f_2. Disciplines 1 and 2 are coupled by the coupling variables, $\boldsymbol{y}_{12}$ and $\boldsymbol{y}_{21}$. They are the output from the corresponding analysis of disciplines 1 and 2, and needed in calculating the objective and constraint functions of disciplines 2 and 1, respectively. In this thesis, we focus on MDO problems that are coupled through only shared design variables, i.e., quasi-separable MDO problems, which are frequently encountered in MDO and the formulation for which is given in Eq. (2.2).

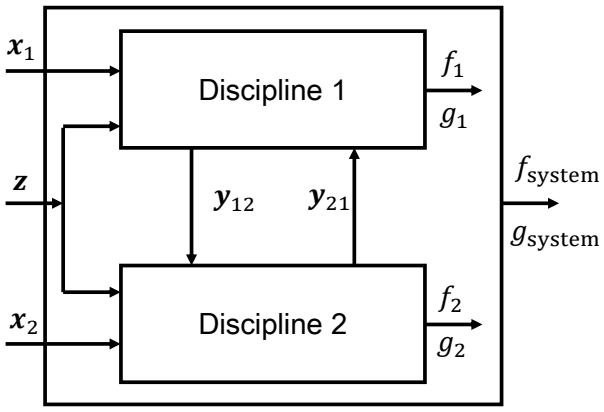


Figure 2.1: A two-disciplinary system.

2.5.1 Existing MDO Methods

A MDO problem can be solved as a single optimisation problem. However, this method requires a high integration of different disciplines, which is impractical in some cases, e.g., when each disciplinary analysis is developed under a different specialised computer code and it is hard to combine these codes [123]. To better handle MDO problems in such situations, researchers have developed distributed optimisation methods based on decomposition, which allow disciplinary autonomy. Since EAs can solve non-differentiable and multimodal optimisation problems, they are more and more widely employed as optimisers in MDO. Existing EA-based distributed optimisation methods for MDO include the MORDACE method (multidisciplinary optimisation and robust design approaches applied to concurrent engineering) [45], the COSMOS method (collaborative optimisation strategy for multi-objective systems) [106], and the ATC method (analytical target cascading) with the multi-objective formulation [89]. The following paragraphs will explain each of these methods in more details.

In the MORDACE method [45], each disciplinary optimisation is performed independently. When disciplinary optimisation is finished, the MORDACE method employs a compromise method on the common decision variables. As changes in common variable values due to compromise will make each disciplinary performance worse, a robust design approach is employed in each disciplinary optimisation to alleviate this. In the robust design approach, in addition to disciplinary objectives, disciplinary optimisation also aims to minimise the sensitivity of performance values to variations of common variables.

The COSMOS method [106] tries to solve multi-objective problems in a multidisciplinary context. It uses a nested decomposition and includes two-level optimisation: supervisor-level and disciplinary-level optimisation. The supervisor level optimises the common variables and provides the values of common variables to the disciplinary level. The disciplinary level optimises the disciplinary variables based on the given common variable values. It then returns the best function value to the supervisor level, which will then be used as the performance value of the given common variables. It can be seen

that the optimisation in the disciplinary level aims to calculate the fitness of the given common variable values, and thus is a nested optimisation. In the COSMOS method, the supervisor level employs a MOGA to search the best common variable values based on the performance values returned from the disciplinary-level optimisation.

The ATC method [89] is a hierarchical multi-level methodology. It propagates system targets through a hierarchical structure and minimises the unattainability when the targets are unattainable [82]. When using ATC to solve the MDO problem in Fig. 2.1, copies of common design variables and coupling variables as well as consistency constraints are created to make the common variables and coupling variables consistent in the final design. ATC uses a penalty function to deal with the consistency constraints. Fig. 2.2 illustrates the decomposition and coordination of ATC in solving the problem in Fig. 2.1.

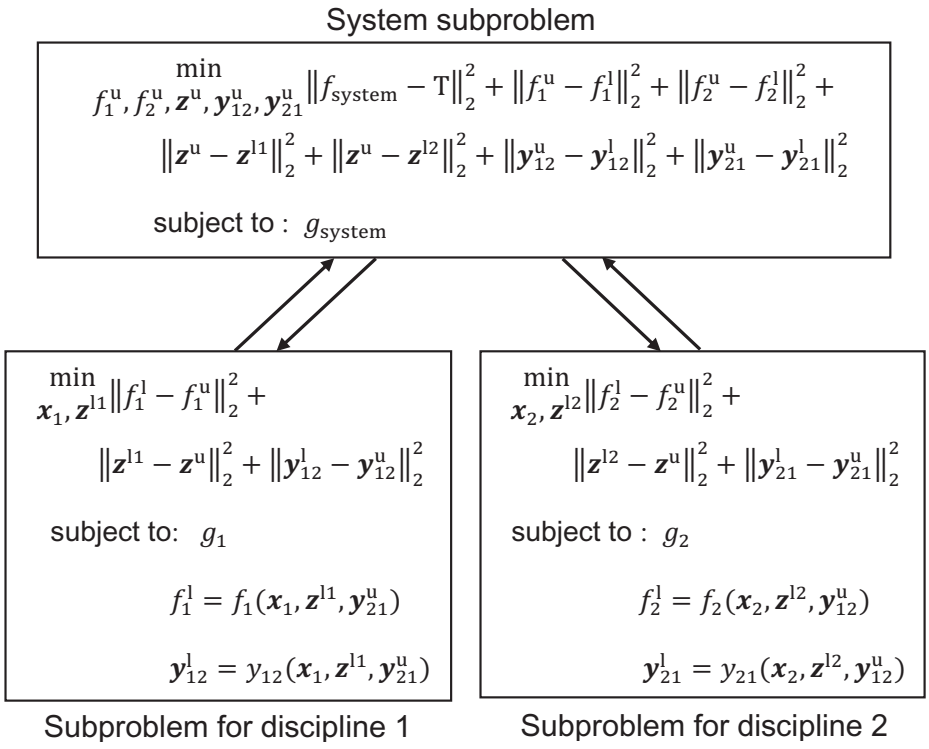


Figure 2.2: ATC decomposition and coordination.

In this figure, T denotes the design targets. The $\boldsymbol{z}^{\mathrm{u}}$ are the duplicate common variables for system optimisation; $\boldsymbol{z}^{\mathrm{l1}}$ and $\boldsymbol{z}^{\mathrm{l2}}$ are for disciplinary optimisation ($\boldsymbol{z}^{\mathrm{u}} = \boldsymbol{z}^{\mathrm{l1}} = \boldsymbol{z}^{\mathrm{l2}}$ at optimality). The $\boldsymbol{y}_{12}^{\mathrm{u}}$ and $\boldsymbol{y}_{21}^{\mathrm{u}}$, and $\boldsymbol{y}_{12}^{\mathrm{l}}$ and $\boldsymbol{y}_{21}^{\mathrm{l}}$ are duplicate coupling variables for system and disciplinary optimisation, respectively ($\boldsymbol{y}_{12}^{\mathrm{u}} = \boldsymbol{y}_{12}^{\mathrm{l}}$ and $\boldsymbol{y}_{21}^{u} = \boldsymbol{y}_{21}^{\mathrm{l}}$ at optimality). The

system-level and discipline-level optimisations are carried out in turn in a loop. When the system-level optimisation is completed, the optimal f_1^{u}, f_2^{u}, $\boldsymbol{z}^{\mathrm{u}}$, $\boldsymbol{y}_{12}^{\mathrm{u}}$ and $\boldsymbol{y}_{21}^{\mathrm{u}}$ are passed to the discipline-level optimisation, which will pass the optimal f_1^{l}, f_2^{l}, $\boldsymbol{z}^{\mathrm{l1}}$, $\boldsymbol{z}^{\mathrm{l2}}$, $\boldsymbol{y}_{12}^{\mathrm{l}}$ and $\boldsymbol{y}_{21}^{\mathrm{l}}$ to the system-level optimisation. Then, another loop begins. In the study in [89], the penalty function in the system optimisation in ATC was transformed into a multi-objective formulation as shown in Eq. (2.11), and the NSGA-II [31] was applied to solve the multi-objective problem.

$$
\begin{aligned}
&\min \ \|f_{\mathrm{system}} - \mathrm{T}\|_2^2 \\
&\min \ \|f_1^{\mathrm{u}} - f_1^{\mathrm{l}}\|_2^2 + \|f_2^{\mathrm{u}} - f_2^{\mathrm{l}}\|_2^2 \\
&\min : \\
&\|\boldsymbol{z}^{\mathrm{u}} - \boldsymbol{z}^{\mathrm{l1}}\|_2^2 + \|\boldsymbol{z}^{\mathrm{u}} - \boldsymbol{z}^{\mathrm{l2}}\|_2^2 + \|\boldsymbol{y}_{12}^{\mathrm{u}} - \boldsymbol{y}_{12}^{\mathrm{l}}\|_2^2 + \|\boldsymbol{y}_{21}^{\mathrm{u}} - \boldsymbol{y}_{21}^{\mathrm{l}}\|_2^2 \\
&\text{suject to} : g_{\mathrm{system}}
\end{aligned}
\tag{2.11}
$$

A cooperative co-evolutionary algorithm was employed to address MDO problems [90, 17, 18, 127]. In these studies, the common variables of an MDO problem are decomposed into disjoint sets and assigned to different disciplines. The coupling variables are handled with duplicate variables or using the implicit iteration strategy [90]. These methods need to make decisions on how the common variables should be decomposed, but such methods have not been developed yet. In [91], the authors suggested decomposing common variables based on their main effects on the disciplinary objectives via orthogonal array-based experimental design before starting optimisation.

2.5.2 Discussions

The aforementioned MDO methods have the following disadvantages. First, the MORDACE method needs to spend a lot of function evaluations in calculating the sensitivity of performance values to variations of common variables. Second, for each common vari-

able value given by the supervisor level, the COSMOS method needs to restart a new disciplinary-level optimisation. This slows down the concurrent design process. The ATC method uses a point-based iterative process, it can easily converge to local optima and is highly sensitive to the numerical inaccuracy of solutions communicated among the subsystems [124]. The existing CooC methods designed for MDO problems need to decompose the common variables but a good decomposition might not exist or needs very costly experimental design to find out. Thus, efficient MDO methods are still needed in the field of CE.

Through decomposition, the original design problem in MDO is divided into several smaller sub-problems, which can be solved in a concurrent fashion. But, as the sub-problems are usually dependent on each other, they can not be solved in isolation but need to collaborate with each other. In the collaboration, a timely information exchange is always needed between different sub-problem optimisation. However, communication usually has a cost and might be different for different problems in practice. As a result, when to exchange information is not easy to determine. In the field of CE, how to set the communication frequency among different subtasks is a very hot topic [62, 74, 75, 108, 131] but the communication frequency issue among different disciplinary designs has not been studied in the literature.

2.6 Dynamic Constrained Optimisation

2.6.1 Dynamic Optimisation Methods

Dynamic constrained optimisation problems belong to the domain of dynamic optimisation problems, which are composed of dynamic unbounded and dynamic constrained optimisation problems. The challenges of dynamic optimisation problems consist in two aspects. Firstly, restarting the optimisation after a change happens often needs long time to find the new optimum and thus is impractical. Secondly, continuing search on the cur-

rent population without any change is inefficient as the current population might partially converge and thus cannot react quickly to changes.

Considering these, researchers have proposed different approaches to handle dynamic optimisation problems. Generally, these approaches can be grouped into three categories: diversity-driven approaches, memory approaches and prediction approaches. Diversity-driven approaches mainly include introducing diversity after a change happens, maintaining diversity during search, and multi-population approaches that maintain multiple populations to locate and track multiple optima simultaneously [93]. Memory approaches store and use good-performing solutions from previous search. They are effective for cyclic environment. Prediction approaches predict new optimal solutions and use the predicted optimal solutions for initialisation once the problem changes. They are suitable for predictable problems. Although a lot of studies have been done in the field of dynamic optimisation, most of them aim at dynamic unbounded or dynamic constrained optimisation problems with bound constraints. As the majority of dynamic optimisation problems in real world are dynamic constrained optimisation problems (DCOPs) and DCOPs have their own characteristics, DCOPs attract researchers' attentions recently.

2.6.2 Dynamic Constrained Optimisation Methods

Dynamic constrained optimisation problems have constraints that are different from bound constraints, and the changes can happen on their objectives, constraints or both. Therefore, the distribution of feasible/infeasible regions in a DCOP might change during the optimisation process, and the global optimum might appear in a newly feasible area without changing the existing optimum or switch from one feasible region to another disconnected one [93].

These raise new challenges. It has been shown in [93] that simple diversity maintaining/introducing strategies (e.g. random-immigrants (RI) [50] and hyper-mutation (HyperM) [22]) become less effective when combined with penalty functions that prefer feasible solutions to infeasible ones. The reasons are as follows. Although RI and Hy-

perM introduce/generate some random solutions, most of them are likely rejected by the used penalty function as the majority of them are infeasible. When faced with DCOPs in which the global optimum moves from one feasible region to another disconnected one, the RI and HyperM method can not perform effectively as this kind of problems need an infeasible path connecting two disconnected feasible regions.

To address this problem, the authors in [94] suggested using constraint handling techniques that can maintain diversified infeasible solutions or tracking the moving feasible regions instead of tracking the moving existing optima. Existing dynamic constrained optimisation methods are mainly conducted along this direction. In addition, dynamic optimisation methods like memory methods and prediction methods have also been modified to address DCOPs. In this section, the existing dynamic constrained optimisation methods are categorised into three classes: diversity-driven approaches, memory approaches and prediction approaches. Each of them will be introduced in the following part of this section.

Diversity-driven Approaches

To maintain diversified infeasible solutions, researchers have used constraint handling techniques that accept both feasible and infeasible individuals. In [92], the authors used the repair method [112] along with RI and HyperM, respectively. For an infeasible individual, the repair method first repairs it to obtain a feasible individual and then uses the fitness of the obtained feasible individual as the fitness of the infeasible individual. In this case, infeasible solutions that are able to make good feasible solutions can survive in the selection process. Similar repair methods were also used in [96] and [5]. However, as the repair method needs to repair every infeasible solution to a feasible one, it needs to check the feasibility of every repaired solution and thus usually requires a lot of feasibility checkings. As a result, the use of repair methods is not a good choice when the feasible regions of the problem are very small or the feasibility checking is very costly.

Alternatively, the authors in [15] employed a simple ranking scheme proposed in [54].

In this scheme, an individual is ranked with respect to each of the objective function values, the sum of constraint violation and the number of violated constraints. The sum of all three ranks for an individual is used to make a comparison when there exist both feasible and infeasible individuals in the population. Through the use of this scheme, those infeasible solutions with a small degree of constraint violation and a small objective function value can be reserved. To maintain population diversity, the authors in [15] also proposed to switch between a global search operator and a local search one according to whether the diversity degree is larger than a threshold. In [16], this work was further improved by using a more complex function of the three ranks to evaluate individuals and employing Shannon's index of diversity as a factor to balance the influence of global-best and local-best search directions.

As mentioned in Section 2.3.1, niching EAs have been widely applied to track moving optima in addressing DOPs [69, 12, 86, 100]. However, little work has been done along the direction of tracking moving feasible regions before the year of 2016. To locate and track feasible regions, this thesis proposed a speciation-based method, called speciated evolution with local search (SELS) [78]. The SELS method utilises deterministic crowding [80] to make comparisons among similar individuals and assortative mating [28] to induce speciation. By doing these, good solutions can be maintained in different feasible regions and thus feasible regions can be located and tracked. More details can be found in Chapter 4. Later, the authors in [13] specifically used multiple sub-populations to locate and track multiple feasible regions. The whole population is divided into multiple sub-populations which are evolved simultaneously. The authors also employed a gradient-based repair method to accelerate the location of feasible regions and adaptive local search to exploit found feasible regions.

Memory Approaches

The authors in [107] first proposed to use a memory approach for DCOPs and modified the abstract memory method to make it fit for DCOPs. In this work, two memories were

built to represent the distribution of previously good individuals and feasible regions. When using these two memories, two schemes, blending and censoring were considered. In the blending scheme, some solutions are generated from each of the two memories and then all of them are introduced into the population. In the censoring scheme, solutions of more than the original size are generated from the memory for the good individuals and then are censored by the memory for the feasible regions by selecting solutions with the largest probability to be feasible. Only the selected solutions are introduced into the population. The experimental results in [107] showed that the modified memory approach with the censoring scheme can help to improve the performance for DCOPs with dynamic fitness functions and static constraints.

To address DCOPs, the authors in [13] also introduced a memory strategy to track previously feasible regions. This strategy maintains two memories. One memory archives previously found local optima and the other saves previously found best solutions. Both memories are divided into species. In the re-initialisation process, a random particle from each species is to be retrieved so as to guarantee the diversity of retrieved individuals.

Prediction Approaches

In [37], the authors combined the infeasibility driven evolutionary algorithm (IDEA) with a prediction method to solve DCOPs. The IDEA is an algorithm designed to deal with static constrained optimisation problems. To approach the constraint boundary from both feasible and infeasible sides, IDEA maintains a certain fraction of infeasible solutions in the population by selecting the best part from both of feasible solutions and infeasible solutions to enter the next generation. The prediction method employed in [37] assumes that the change patterns of spatial optima locations can be fitted by an AutoRegressive model. Under this assumption, the AutoRegressive Integrated Moving Average method was applied to predict the location of future optima based on the obtained best individuals from previous generation. Solutions generated from the prediction method will be used to replace individuals in the current population. This work was further improved in [38]

by reducing the use of memory and using a new anticipation mechanism.

Different from predicting locations of future optima, the work in [13] introduced a species-based prediction method to predict the locations of future feasible regions. Each species is considered as a feasible region and the prediction method uses the species seed as a representative of the species. Based on the species seeds from past generations, this work aims to predict locations of multiple feasible regions in parallel under the assumption that each feasible region moves linearly. Concretely, suppose that $Mem(t-1)$ denotes the species seeds at time $t-1$ and $Mem(t)$ denotes the species seeds at time t. For each cm in $Mem(t)$, this method considers the nearest particle pm in $Mem(t-1)$ to cm as from the same moving feasible region. Then, for each pair of $< pm, cm >$, the predicted location of the same feasible region at time $t+1$ is calculated as $cm + (pm - cm)$. These predicted locations will be used to replace the individuals in the original population once a change is detected.

2.6.3 Discussions

As introduced above, some dynamic constrained optimisation methods have already been developed for DCOPs. However, the use of repair methods to accept diversified infeasible solutions usually needs a lot of feasibility checkings. The use of constraint handling techniques that accept both feasible solutions and infeasible solutions might not react quickly when the global optimum moves to another disconnected feasible region especially when the feasible region in which the new best solution located is far away. Memory approaches and prediction approaches are not appropriate when the DCOPs are not cyclic and predictable. Therefore, efficient dynamic constrained optimisation methods are still required for DCOPs.

Furthermore, in the field of dynamic optimisation, DOPs with very fast changes are the most challenging problems. When the changes happen very fast, the existing diversity-driven approaches, memory approaches and prediction approaches can not work well. The diversity-driven approaches do not have enough time to find a satisfactory solution. As

the time for optimisation is very short, the memory approaches can not have optimal solutions to archive and thus the prediction approaches can not have useful samples for accurate prediction. However, none of the above-mentioned methods aimed to deal with fast-changing DCOPs.

2.7 Chapter Summary

In this chapter, the background knowledge of EAs and multi-species EAs have been given. Three types of multi-species EAs that are employed in this thesis work are introduced in detail as well as what types of optimisation problems each of them aims to solve. The existing co-evolution issues including individual evaluation and communication frequency in CooC are discussed. The research gaps in addressing CE problems with EAs and the existing dynamic constrained optimisation are analysed.

CHAPTER 3

COOPERATIVE CO-EVOLUTION FOR DESIGN OPTIMISATION IN CONCURRENT ENGINEERING

In Chapter 2, different types of multi-species EAs (i.e. multi-population EAs, cooperative co-evolution (CooC), and competitive co-evolution (ComC)) and the types of optimisation problems they are suitable to solve were reviewed. In this chapter, we present two new concurrent design methods based on parallel CooC to address the two types of concurrent engineering (CE) problems defined in Section 2.1 in Chapter 2. Before introducing the proposed co-evolutionary concurrent design methods, in Section 3.1, we first introduce the motivation of this work. Then, Section 3.2 will detail the proposed co-evolutionary methods. In Section 3.3, the comparison results between the proposed methods and existing methods will be presented on the universal electric motor (UEM) design problems and a commonly used MDO test problem. Then, the best communication frequencies between different sub-populations in the proposed methods with/without communication costs will be studied. Section 3.4 will describe a self-adaptive method for adapting the best communication frequencies and report the comparison results. Finally, Section 3.5 will summarise this chapter and give conclusions.

3.1 Motivation

As a well-known engineering practice, concurrent engineering (CE) considers all elements involved in a product's life cycle (from functionality, producibility to maintenance issues, and finally disposal and recycling) from the early stages of product development, and advocates executing all design tasks in parallel or with some overlaps [63, 130]. This contrasts with traditional sequential engineering that conducts different tasks separately and at different times. Consequently, in CE, errors can be discovered earlier when the design is still flexible, and laborious re-designs can thus be avoided [2, 131].

However, CE also increases the complexity of the design problem. Since it considers all parts or all aspects of the product design simultaneously, there exist two types of complex optimistion problems in the field of CE. The first type of design optimisation problem involves many design variables and the second type requires different disciplinary expertise to solve. To enable concurrent design, decomposition-based optimisation strategies are preferred in CE. The original problem is usually divided into small sub-problems so that each sub-problem can be solved individually and simultaneously.

In the literature, there are three popular decomposition methods in CE, i.e., product decomposition, process decomposition and problem decomposition [64]. Product decomposition depends on the predefined product structure, and problem decomposition is based on the dependencies between design parameters. Process decomposition is often applied in multi-disciplinary design optimization (MDO) problems where different disciplines share design variables and are usually coupled by coupling variables. To retain disciplinary autonomy, process decomposition methods break up the design problem according to engineering disciplines.

As discussed in Section 2.5 in Chapter 2, CE could benefit from evolutionary computation techniques in many aspects. For instance, when CE involves sub-problems with non-convex or non-differentiable objective functions, EAs can be used as the sub-problem solver in the framework of collaborative optimisation [33, 132] and ATC [89, 133]. Moreover, the divide-and-conquer idea behind CE is essentially similar to that of CooC which

decomposes the decision variables into several disjoint sub-components and evolves each sub-component with an EA. Thus, CooC can be directly applied to CE problems in which decomposition is based on the predefined product structure and there are no shared design variables between different sub-problems. In the literature, CooC has also been employed in the field of CE to deal with MDO problems [17, 18, 90, 127].

However, some issues arise when using CooC to address CE problems. As discussed in Section 1.3.1 in Chapter 1, first, CooC can easily converge to mediocre stable states [49]. Although researchers have proposed different individual evaluation methods to alleviate this situation as reviewed in Section 2.4.1, the existing methods are computationally inefficient and restrict the application of CooC in CE as discussed in Section 2.4.1. Second, how to handle the shared design variables when applying CooC for MDO problems are not well dealt with. Third, communication frequency between different sub-populations can have a big performance effect on the performance of CooC. But there are few studies conducted to answer how to set the communication frequency in CooC especially when communication incurs a cost in a concurrent design environment.

Motivated by the potential of CooC to CE, this chapter aims to investigate CooC-based design optimisation methods in the context of CE. Concretely, this chapter will try to address the above-mentioned issues through using CooC to address the two types of CE problems defined in Section 2.1.1 in Chapter 2, and answer the questions of *how to better evaluate individuals in each sub-population when using CooC to address the first type of CE problem*, *how to better handle the shared design variables when using CooC to address the second type of CE problem*, and *how often different sub-populations should communicate with each other in the CooC-based concurrent design optimisation* as posed in Section 1.3.1 in Chapter 1.

To answer these questions, for the first type of CE problems in which the decomposition of product is assumed to be known in prior, this chapter will present a new novelty-driven CooC which uses a new and computationally efficient novelty calculation method, and employs a stochastic selection process with an adaptive probability to adjust the trade-off

between novelty and fitness. For the second type of CE problems (i.e. quasi-separable MDO problems), instead of evolving the shared variables in a single sub-problem, this chapter will introduce a novel co-evolutionary concurrent design method in which the shared variables are handled through the use of duplicates and consistency constraints, and the stochastic ranking method [111] with an adaptive probability is employed to deal with the constraints. The performance of the two proposed methods is demonstrated through comparisons to the method that optimises all parts (disciplines) as a whole as well as other state-of-the-art methods on universal electric motor (UEM) design problems and a commonly used MDO test problem. Additionally, this chapter will introduce a self-adaptive method to adapt the communication frequency during the optimisation process after studying how the communication frequency among sub-populations affects the performance of the proposed CooC methods.

3.2 The Proposed Methods

The two proposed co-evolutionary concurrent methods are based on the framework of parallel CooC. Fig. 3.1 shows a parallel CooC optimisation process. In parallel CooC, all sub-populations are evolved simultaneously. The optimisation process of CooC is divided into cycles. At the end of each cycle, different sub-populations communicate with each other through collaborative individuals in their current populations, which are used to evaluate the fitness of the individuals in the other sub-populations in the next cycle. In the literature, there exist different methods to select collaborative individuals [119] and the most commonly used method is to choose the best individual in the concurrent population. In Fig. 3.1, the decision variables are decomposed into two sub-components ($\boldsymbol{x}_1$ and $\boldsymbol{x}_2$). f denotes the objective function. Each of t_1, t_2, t_3 denotes a communication time point between two sub-populations (i.e., Pop_1 and Pop_2).

In the following part of this section, we will first introduce a new novelty-driven CooC method with stochastic selection (NDCC-SS for short) for the mono-disciplinary design

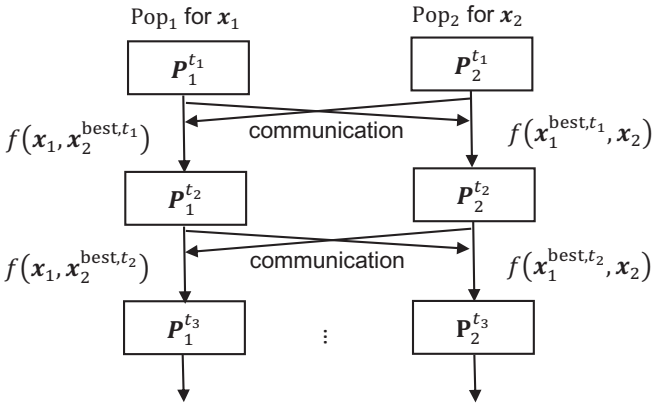


Figure 3.1: Parallel cooperative co-evolutionary optimisation process.

problems in Eq.(2.1), and then a novel co-evolutionary concurrent design method (CCDM for short) for the quasi-separable MDO problems in Eq.(2.2).

3.2.1 A New Novelty-Driven Cooperative Co-evolution with Stochastic Selection (NDCC-SS)

As reviewed in Section 2.4.1 in Chapter 2, novelty-driven CooC selects individuals based not only on the fitness but also on the novelty score of individuals. In the proposed NDCC-SS method, we hope to prevent CooC from converging to suboptimal states by maintaining the diversity of the selected individuals in the selection process because the search direction of CooC depends on the selected individuals. Motivated by this, we use a new novelty metric in NDCC-SS that measures individuals' novelty score based only on the individuals that are already selected for generating offsprings. Therefore, novelty calculation and EA selection are conducted together in NDCC-SS. For the sake of computational efficiency, the novelty score of one individual, $\boldsymbol{x}$, is evaluated as follows:

$$nov(\boldsymbol{x}) = dist(\boldsymbol{x}, \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{x}_{\mathrm{s}}^{i}) \tag{3.1}$$

where $\boldsymbol{x}_{\mathrm{s}}^{i}$ denotes the i-th already selected individual and k denotes the number of the individuals already selected. Moreover, the binary tournament selection is employed and

Algorithm 5 describes the selection process in NDCC-SS. Note the cost of the novelty calculation process in Algorithm 5 increases linearly with population size, which has a time complexity of $\mathcal{O}(NP * \text{complexity}(dist))$ (NP denotes the population size). Thus, this new novelty calculation is more computationally efficient than the method in Eq. (2.8) which has a complexity of $\mathcal{O}(NP^2 * \text{complexity}(dist) + NP^2 log(NP))$ even without considering archived individuals.

Algorithm 5 Novelty Calculation and Binary Tournament Selection

```
Select the best individual in the current population as x_s^1
Set c = x_s^1 and k = 1
while k ≤ NP − 1 do
    Randomly pick two individuals from the current population, x_1, x_2
    Calculate the fitness of them and set the novelty of x_i(i = 1, 2) as dist(x_i, c)
    Set k = k + 1 and select the winner of x_1 and x_2 as x_s^k
    Set c = (c * (k − 1) + x_s^k)/k
end while
Let all x_s^i(i = 1, 2, ...NP) enter next generation
```

In addition, we use a stochastic selection process with an adaptive probability to make a trade-off between novelty and fitness objectives. Algorithm 6 shows the comparison process between two individuals, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. The novelty and fitness score of $\boldsymbol{x}_i(i = 1, 2)$ is $nov(\boldsymbol{x}_i)$ and $f(\boldsymbol{x}_i)$, respectively. The parameter p_n denotes the probability of selecting individuals based on their novelty scores, and rand$(0, 1)$ is a random number generated from a uniform distribution on [0, 1]. The value of p_n in the new method is linearly decreased from p_0 to p_f within a specified number of generations and is kept constant as p_f after that. In detail, p_n^G is calculated as follows:

$$\begin{cases} p_n^G = p_0 - (p_0 - p_f) * \frac{G}{r * MaxGen}, & if\ G \leq r * MaxGen \\ p_f, & otherwise \end{cases} \tag{3.2}$$

where G is the current number of generations, r is a real number ($0 \leq r \leq 1$), and $MaxGen$ denotes the maximum number of generations.

Through the use of stochastic selection with an adaptive probability, NDCC-SS has

Algorithm 6 Comparison($\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $f(\boldsymbol{x}_1)$, $nov(\boldsymbol{x}_1)$, $f(\boldsymbol{x}_2)$, $nov(\boldsymbol{x}_2)$, p_n)

```
if rand(0, 1) < p_n then
    if nov(x_1) > nov(x_2) then
        x_1 wins
    else
        x_2 wins
    end if
else
    if f(x_1) > f(x_2) then
        x_1 wins
    else
        x_2 wins
    end if
end if
```

two advantages in balancing the novelty and fitness objectives compared with the weighted sum method and multi-objective method as introduced in Section 2.4.1 in Chapter 2. One is that the stochastic selection shows a direct and explicit scheme to determine how many comparisons are dominated by the novelty or fitness objectives, while the importance of novelty or fitness is unclear in the weighted sum method. The other is that our adaptive scheme changes the emphasis on the two objectives as the evolution proceeds. Through the linear decrease of p_n, it is expected that the new method will focus more on exploration in the early stage of search and on exploitation in the later stage.

To address the mono-disciplinary design problems as defined in Eq. (2.1) in Chapter 2, the parallel CooC framework in Fig. 3.1 with this new novelty-driven method is used in our work. Assume a product has two parts ($\boldsymbol{x}_1$ and $\boldsymbol{x}_2$). In the beginning, an initial sub-population for each of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ is randomly generated. At each communication step (e.g., t_1, t_2, t_3), two sub-populations communicate with each other the best individual in their current populations. Then, $\boldsymbol{x}_1$ is optimised using an EA for a fixed number of generations (until the next communication step) using the new $\boldsymbol{x}_2$ in fitness evaluation, meanwhile $\boldsymbol{x}_2$ is optimised using an EA with the new $\boldsymbol{x}_1$ in fitness evaluation. In each EA optimisation, individuals in each sub-population are selected based on either novelty (calculated using Eq. (3.1)) or fitness scores according to Algorithm 6.

3.2.2 A Novel Co-evolutionary Concurrent Design Method (CCDM) for Quasi-Separable MDO Problems

For quasi-separable MDO problems as defined in Eq. (2.2) in Chapter 2, the original problem is decomposed along disciplines and duplicate variables are introduced to deal with the common design variables $\boldsymbol{z}$ in CCDM. The design is assumed to involve two disciplines and each has local design parameters $\boldsymbol{x}_i (i = 1, 2)$, respectively. Through the use of duplicate variables, $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$, the original problem in Eq. (2.2) can be decomposed as follows:

1. sub-problem 1

$$\begin{aligned} &\min_{\boldsymbol{x}_1, \boldsymbol{z}^1} \ f_1(\boldsymbol{x}_1, \boldsymbol{z}^1) \\ &\text{subject to} : \boldsymbol{z}^1 = \boldsymbol{z}^2 \end{aligned} \tag{3.3}$$

2. sub-problem 2

$$\begin{aligned} &\min_{\boldsymbol{x}_2, \boldsymbol{z}^2} \ f_2(\boldsymbol{x}_2, \boldsymbol{z}^2) \\ &\text{subject to} : \boldsymbol{z}^2 = \boldsymbol{z}^1 \end{aligned} \tag{3.4}$$

After the decomposition, the CCDM optimises each sub-problem in a sub-population with a specified EA simultaneously, as shown in Fig. 3.1. At each communication step, they communicate with each other their best design information (i.e., $\boldsymbol{z}^i$ $(i = 1, 2)$) in the current population in order to evaluate their separate designs.

To deal with the equality constraints in Eqs.(3.3) and (3.4), CCDM transforms them into an inequality constraint as follows:

$$\frac{\sum_{k=1}^{nc} |z_k^1 - z_k^2|}{nc} \leq \delta \tag{3.5}$$

where nc is the number of common variables, z_k^1 and z_k^2 are the k-th element of $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$, respectively; δ is the tolerance parameter, and an adaptive setting [81] is applied here. In

this adaptive setting, δ is adapted by using the following expression:

$$\delta(t+1) = \frac{\delta(t)}{\hat{\delta}} \tag{3.6}$$

where $\delta(0)$ is set as the median of equality constraint violations (i.e., $\frac{\sum_{k=1}^{nc} |z_k^1 - z_k^2|}{nc}$) over all individuals in the initial population, $\hat{\delta}$ is the rate of decay, which is set in such a way that the δ value will decrease to a final value δ_f after a specified number of generations and will be kept as δ_f after that.

To deal with the generated inequality constraints and other existing inequality constraints, a state-of-the-art constraint handling method, the stochastic ranking method [111] is used in CCDM. In the stochastic ranking method, when making comparisons among feasible individuals, the individual with better objective value is selected. Otherwise, the comparison is made based on the objective values of individuals with the probability P_f and based on the constraint violation values with the probability $(1 - P_f)$. The constraint violation for the inequality constraint in Eq. (3.5) is defined as:

$$\phi(\boldsymbol{z}) = \begin{cases} 0 & \text{if } \frac{\sum_{k=1}^{nc} |z_k^1 - z_k^2|}{nc} \leq \delta \\ \delta - \frac{\sum_{k=1}^{nc} |z_k^1 - z_k^2|}{nc} & \text{otherwise} \end{cases} \tag{3.7}$$

In our work, the probability parameter, P_f, is decreased linearly from $P_f = 0.475$ in the initial generation to $P_f = 0.25$ in the final generation. After the optimisation, the mean values of $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$ in the best solutions are used as the final values of $\boldsymbol{z}$ in CCDM.

In both NDCC-SS and CCDM, the communication interval denotes the interval between the generation numbers of two sequential communications in the parallel CooC as shown in Fig. 3.1. There will be more than two sub-populations if more than two parts (disciplines) exist in the design, but the whole framework remains unchanged.

3.3 Experimental Studies

3.3.1 Case Studies

To evaluate the efficacy of the two proposed methods, universal electric motor (UEM) design problems and a geometric programming problem are used as case studies. The UEM design problem is a widely used benchmark problem in the field of product family design [116, 117, 59]. In this study, two scenarios of the UEM design problems are considered.

The Baseline UEM Design Problem

One UEM has 8 design parameters, $\{N_c, N_s, A_{wf}, A_{wa}, I, r_o, t, L\}$ [43]. The description of each parameter can be found in [117]. The design of an electric motor can be formulated as a two-objective constrained optimisation problem, given as follows:

$$\begin{aligned}
&\min_{\boldsymbol{x}} \{Mass(\boldsymbol{x}), -\eta(\boldsymbol{x})\} \\
&\text{where :} \\
&\boldsymbol{x} = \{N_c, N_s, A_{wf}, A_{wa}, I, r_o, t, L\} \\
&\text{subject to :} \begin{cases} H(\boldsymbol{x}) \leq 5000[\text{Amp} \cdot \text{turns/m}] \\ \frac{r_o}{t} \geq 1 \\ Mass(\boldsymbol{x}) \leq 2[\text{kg}] \\ \eta(\boldsymbol{x}) \geq 15\% \\ P(\boldsymbol{x}) = 300[\text{W}] \\ T(\boldsymbol{x}) = a[\text{Nm}] \end{cases}
\end{aligned} \tag{3.8}$$

where $Mass(\boldsymbol{x})$ and $\eta(\boldsymbol{x})$ denote the weight and efficiency calculation function, respectively. In Eq. (3.8), a denotes the users' requirement of the torque output ($T(\boldsymbol{x})$), which is a real number and specified a priori. In this chapter, for each design $\boldsymbol{x}$, the penalty with

respect to the constraints in Eq. (3.8) is calculated and added to $\{Mass(\boldsymbol{x}), -\eta(\boldsymbol{x})\}$. The resulting $\{Mass(\boldsymbol{x}), -\eta(\boldsymbol{x})\}$ are used as the final objective values of the design. Details about the penalty calculation and the formulations of $H(\boldsymbol{x})$, $Mass(\boldsymbol{x})$, $\eta(\boldsymbol{x})$, $P(\boldsymbol{x})$ and $T(\boldsymbol{x})$ can be found in the Appendix.

For the UEM design problem, if the preference between $Mass(\boldsymbol{x})$ and $-\eta(\boldsymbol{x})$ is known, a weighted sum of these two objective functions can be defined as the following expression:

$$f_{\text{weighted}}(\boldsymbol{x}) = w_1 * (1 - \eta(\boldsymbol{x})) + w_2 * Mass_{\text{normalised}}(\boldsymbol{x}) \tag{3.9}$$

where $Mass_{\text{normalised}}$ represents the normalised weight derived by dividing the original weight by the maximum allowable weight: $Mass_{\max} = 2$ [kg]. w_1 and w_2 are weight coefficients that are assumed to be equal (i.e., $w_1 = w_2 = 0.5$) in [3]. Through the weighted sum, the UEM design problem is transformed into a single-objective optimisation problem.

Scenario 1: Single Motor Design

This scenario aims to design a motor with respect to one torque requirement a (Eq. (3.8)). We assume for this scenario that 8 design parameters are decomposed into two groups $\{N_c, A_{wf}, I, t\}$ and $\{N_s, A_{wa}, r_o, L\}$ in the beginning. This scenario is used as a case study for the mono-disciplinary design problems in this chapter. For this scenario, the single-objective case is considered. In this case, we use the weighted sum function in Eq. (3.9) as the goal function, and set both w_1 and w_2 to 0.5. The range for each design parameter of each UEM motor is set the same as in [117], which is given in Table 3.1. Two design problems are considered by using different a values in this chapter. One is $a = 0.3$, and the other is $a = 0.5$. When applying the proposed NDCC-SS to this scenario, each group of the design variables of a motor is optimised simultaneously with a different sub-population.

Table 3.1: Range setting for motor parameters

Parameter	Range
N_c	[100, 1500]
N_s	[1, 500]
A_{wf}	[0.01, 1.00] ([mm^2])
A_{wa}	[0.01, 1.00] ([mm^2])
I	[0.1, 6.0] (Amp)
r_o	[1, 10] (cm)
t	[0.5, 100] (mm)
L	[0.1, 20] (cm)

Scenario 2: Overlapping Motors Design

In this scenario, two or more overlapping motors need to be designed, each with a different torque requirement (i.e. a in Eq. (3.8)). Here, overlapping motors refers to different motors sharing some design parameters. This scenario captures some of the important characteristics of designing a product family that has common components in case to reduce production costs. In the literature, the authors in [126] considered designing multiple aircrafts, each with a different mission but sharing the same design of the wing. In this scenario, the overall goal is:

$$\begin{aligned} &\min_{\boldsymbol{x},\boldsymbol{z}} \sum_{i=1}^{m} f_{\text{weighted},i}(\boldsymbol{x}_i, \boldsymbol{z}) \\ &\text{i.e.,}\min_{\boldsymbol{x},\boldsymbol{z}} \sum_{i=1}^{m} \left(w_1 * (1 - \eta(\boldsymbol{x}_i, \boldsymbol{z})) + w_2 * Mass_{\text{normalised}}(\boldsymbol{x}_i, \boldsymbol{z})\right) \end{aligned} \tag{3.10}$$

where m is the number of motors, $\boldsymbol{x}_i$ denotes the local design parameters of the i-th motor, and $\boldsymbol{z}$ denotes the common design parameters. In this scenario, we consider designing 2 motors with different a values in Eq. (3.8) and assume they have 2 parameters (t, L) in common. Three different test problems are studied by using different a values. For the first problem, the a values of the two motors are set to 0.1 and 0.125, respectively. For the second problem, they are set to 0.1 and 0.3, respectively. For the third problem, they are 0.05 and 0.5, respectively. This scenario is used as a case study for the quasi-separable MDO problem and the proposed CCDM method.

A Geometric Programming Problem

The original geometric programming problem [9] is formulated as follows:

$$\min_{z_i \geq 0, i=1,2,\ldots,14} z_1^2 + z_2^2$$

$$\text{subject to :} \begin{cases} z_3^{-2} + z_4^2 - z_5^2 \leq 0 \\ z_5^2 + z_6^{-2} - z_7^2 \leq 0 \\ z_8^2 + z_9^2 - z_{11}^2 \leq 0 \\ z_8^{-2} + z_{10}^2 - z_{11}^2 \leq 0 \\ z_{11}^2 + z_{12}^{-2} - z_{13}^2 \leq 0 \\ z_{11}^2 + z_{12}^2 - z_{14}^2 \leq 0 \\ z_1^2 - z_3^2 - z_4^{-2} - z_5^2 = 0 \\ z_2^2 - z_5^2 - z_6^2 - z_7^2 = 0 \\ z_3^2 - z_8^2 - z_9^{-2} - z_{10}^{-2} - z_{11}^2 = 0 \\ z_6^2 - z_{11}^2 - z_{12}^2 - z_{13}^2 - z_{14}^2 = 0 \end{cases} \tag{3.11}$$

By using the equality constraints, this problem can be transformed into the following formulation:

$$\min_{z_i \geq 0, i=4,5,7,8,\ldots,14} \sum_{i=1}^{2} f_i(\boldsymbol{x}_i, \boldsymbol{z})$$

where :

$$\boldsymbol{x}_1 = \{z_4, z_8, z_9, z_{10}\},\ \boldsymbol{x}_2 = \{z_7, z_{12}, z_{13}, z_{14}\},\ \boldsymbol{z} = \{z_5, z_{11}\}$$

$$f_1(\boldsymbol{x}_1, \boldsymbol{z}) = z_4^{-2} + z_5^2 + z_8^2 + z_9^{-2} + z_{10}^{-2} + z_{11}^2$$

$$f_2(\boldsymbol{x}_2, \boldsymbol{z}) = z_5^2 + z_7^2 + z_{11}^2 + z_{12}^2 + z_{13}^2 + z_{14}^2$$

$$\text{subject to :} \begin{cases} (z_8^2 + z_9^{-2} + z_{10}^{-2} + z_{11}^2)^{-1} + z_4^2 - z_5^2 \leq 0 \\ z_5^2 + (z_{11}^2 + z_{12}^2 + z_{13}^2 + z_{14}^2)^{-1} - z_7^2 \leq 0 \\ z_8^2 + z_9^2 - z_{11}^2 \leq 0 \\ z_8^{-2} + z_{10}^2 - z_{11}^2 \leq 0 \\ z_{11}^2 + z_{12}^{-2} - z_{13}^2 \leq 0 \\ z_{11}^2 + z_{12}^2 - z_{14}^2 \leq 0 \end{cases} \tag{3.12}$$

It can be seen from Eq. (3.12) that the original problem is transformed into a constrained quasi-separable MDO problem with common variables (z_5, z_{11}). This problem is used as another case study for the quasi-separable MDO problems and the proposed CCDM. It is decomposed into two sub-problems with duplicate variables ($(z_{5,1}, z_{5,2})$ and $(z_{11,1}, z_{11,2})$) as follows:

1. sub-problem 1

$$\min_{z_i \geq 0, i=4,8,9,10, z_{5,1} \geq 0, z_{11,1} \geq 0} z_4^{-2} + z_{5,1}^2 + z_8^2 + z_9^{-2} + z_{10}^{-2} + z_{11,1}^2$$

$$\text{subject to :} \begin{cases} (z_8^2 + z_9^{-2} + z_{10}^{-2} + z_{11,1}^2)^{-1} \\ +z_4^2 - z_{5,1}^2 \leq 0 \\ z_8^2 + z_9^2 - z_{11,1}^2 \leq 0 \\ z_8^{-2} + z_{10}^2 - z_{11,1}^2 \leq 0 \\ z_{5,1} - z_{5,2} = 0 \\ z_{11,1} - z_{11,2} = 0 \end{cases} \tag{3.13}$$

2. sub-problem 2

$$\min_{z_i \geq 0, i=7,12,13,14, z_{5,2} \geq 0, z_{11,2} \geq 0} z_{5,2}^2 + z_7^2 + z_{11,2}^2 + z_{12}^2 + z_{13}^2 + z_{14}^2$$

$$\text{subject to :} \begin{cases} z_{5,2}^2 + (z_{11,2}^2 + z_{12}^2 + z_{13}^2 + z_{14}^2)^{-1} - z_7^2 \leq 0 \\ z_{11,2}^2 + z_{12}^{-2} - z_{13}^2 \leq 0 \\ z_{11,2}^2 + z_{12}^2 - z_{14}^2 \leq 0 \\ z_{5,1} - z_{5,2} = 0 \\ z_{11,1} - z_{11,2} = 0 \end{cases} \tag{3.14}$$

3.3.2 Overview on Compared Algorithms

In the experiments, our proposed methods, NDCC-SS and CCDM, are evaluated on the aforementioned case studies, and then comparisons are made between these two methods and some existing methods in terms of the quality of the best solution obtained with a fixed number of function evaluations.

Existing methods used for comparison on Scenario 1 of the UEM design problem include:

- CooC: the original CooC that evaluates individuals based only on their fitness.
- Novelty-driven CooC with the weighted sum method (NDCC-WS) [26]: this method uses a linearly weighted sum of fitness and novelty as given in Eq. (2.9) to evaluate individuals. The novelty metric proposed in [66] was used to calculate the novelty scores of individuals. As given in Eq. (2.8), it calcuates the novelty of an individual based on its nearest neighbours in the current population and optionally past individuals saved in an archive. In the experiments, two different settings for the novelty calculation are applied. One is calculating novelty based only on the individuals in the population. The other is based not only on the individuals in the population but also the archive the saves previous individuals. When using the archive, λ randomly selected individuals in the population are added to the archive at every generation. In the experiments, novelty computation is done with $k = 15$ and $\lambda = 6$ according to [26, 48]. Also, three different settings of the weight ρ in Eq. (2.9) in Chapter 2 are considered: $\rho = 0.25$, $\rho = 0.5$, and $\rho = 0.75$. In total, there are 6 different settings of NDCC-WS used for comparison, i.e., NDCC-WS(0.25), NDCC-WS(0.50), NDCC-WS(0.75), NDCC-WS-a(0.25), NDCC-WS-a(0.50), NDCC-WS-a(0.75) ('a' means with an archive, 0.25, 0.50 and 0.75 mean $\rho = 0.25$, $\rho = 0.5$ and $\rho = 0.75$, respectively).
- Novelty-driven CooC with the multi-objective trade-off method (NDCC-MO) [49]: this method considers the fitness and novelty score of individuals as two objectives, and uses a multi-objective evolutionary algorithm, NSGA-II [31], to evolve indiviudals. Both settings, with and without an archive, are considered. They are noted as NDCC-MO and NDCC-MO-a ('a' means with an archive), respectively. When implementing NDCC-MO and NDCC-MO-a in the experiments, the binary tournament selection is based on the domination between two individuals. In the selection

process, the one that dominates the other is selected. If neither is dominated by the other, one of the two individuals will be chosen randomly.

For Scenario 2, the proposed CCDM is compared with the baseline method that optimises all decision variables as a whole without any decomposition (i.e., addressing the optimisation problem in Eq. (3.10) directly). This is to study whether the decomposition and coordination of the common variables that are used in CCDM really work. We also implemented the COSMOS method (collaborative optimisation strategy for multi-objective systems) [106] that uses nested decomposition on Scenario 2. The resulting method is named collaborative optimisation strategy for single-objective system (COSSOS) in this chapter. COSSOS and CCDM are also compared on Scenario 2. For the geometric programming problem, the solution quality obtained by CCDM is compared with the result achieved by two existing methods:

- AAO (All-At-Once): it optimises all decision variables at once [60],
- ATC-MO: the ATC method with the MO formulation [49].

Table 3.2 summarises the compared methods on each test scenario considered in the experimental study. For all the methods that use the parallel CooC framework, different sub-populations communicate with each other at every generation in the implementation.

Table 3.2: A summary of compared methods on the test problems , here 'a' means with an archive, 0.25, 0.50 and 0.75 mean $\rho = 0.25$, $\rho = 0.5$ and $\rho = 0.75$, respectively.

Problem	Method
Scenario 1	CooC, NDCC-WS(0.25), NDCC-WS(0.50), NDCC-WS(0.75), NDCC-WS-a(0.25), NDCC-WS-a(0.50), NDCC-WS-a(0.75), NDCC-MO, NDCC-MO-a, NDCC-SS
Scenario 2	CCDM, Baseline Method, COSSOS
Geometric programming	CCDM, AAO, ATC-MO

For all the experiments, except for AAO and ATC-MO for which the original results in [60] and [49] are used for comparison, we use a real-coded genetic algorithm with

binary tournament selection and standard operators for simulated binary crossover and polynomial mutation [30] as the underlying EA. The crossover probability is set to 0.9 and the mutation probability is set to $\frac{1}{n}$ (where n is the number of decision variables). The distribution indices [30] for the crossover and mutation operators are set to 15 and 20, respectively. Note that the same EA is used in implementation to allow for a fair comparison.

3.3.3 Comparison Results

Scenario 1: Single Motor Design

For this scenario, the proposed NDCC-SS is compared to CooC, NDCC-WS(0.25), NDCC-WS(0.50), NDCC-WS(0.75), NDCC-WS-a(0.25), NDCC-WS-a(0.50), NDCC-WS-a(0.75), NDCC-MO and NDCC-MO-a on the two single motor design problems. For all these 10 methods, the population size for each sub-population was set to 50, and the maximum number of function evaluations was set to 80000. The parameters of NDCC-SS, p_0, p_f and r, were set to 0.45, 0 and 0.4, respectively. Each method was run for 100 times independently. Then, we used Wilcoxon rank-sum test at a significance level of 0.05 to make comparisons between NDCC-SS and each other method.

Table 3.3 gives the means and standard deviations of the best function values over 100 runs obtained by each method as well as the statistical comparison results between NDCC-SS and each of the other methods on the two test problems. It can be seen from Table 3.3 that the proposed NDCC-SS performed the best among all algorithms on both test problems. The evolutionary curves of CooC, NDCC-WS-a(0.25), NDCC-MO, and NDCC-SS on the single motor design problems with $a = 0.3$ and $a = 0.5$ are given in Figs. 3.2 and 3.3, respectively.

To investigate whether both the new novelty calculation and the adaptive stochastic selection process contribute to the good performance of NDCC-SS, we furthermore conducted comparative experiments. In the first experiment, we fixed the probability

Table 3.3: Comparison between CooC, NDCC-WS, NDCC-MO and NDCC-SS for single motor design problems. The minus sign (−) denotes the method in the corresponding row is statistically worse than NDCC-SS on the test problem in the corresponding column. The best results are marked in **bold**.

Method	$a = 0.3$	$a = 0.5$
CooC	4.90e−01±1.41e−01 −	1.87e+01±6.00e+01 −
NDCC-WS(0.25)	8.73e−01±2.72e+00 −	4.56e+00±1.22e+01 −
NDCC-WS(0.50)	6.67e−01±9.93e−01 −	4.26e+00±1.37e+01 −
NDCC-WS(0.75)	2.93e+00±1.97e+01 −	6.35e+00±2.14e+01 −
NDCC-WS-a(0.25)	5.08e−01±5.45e−01 −	1.00e+00±1.86e+00 −
NDCC-WS-a(0.50)	1.14e+00±1.40e+00 −	1.56e+00±3.94e+01 −
NDCC-WS-a(0.75)	1.20e+01±2.36e+01 −	5.97e+01±9.01e+01 −
NDCC-MO	5.01e−01±4.96e−02 −	1.94e+00±3.55e+00 −
NDCC-MO-a	5.35e−01±6.93e−02 −	1.92e+00±4.38e+00 −
NDCC-SS	**4.43e−01±3.59e−02**	**5.66e−01±3.72e−02**

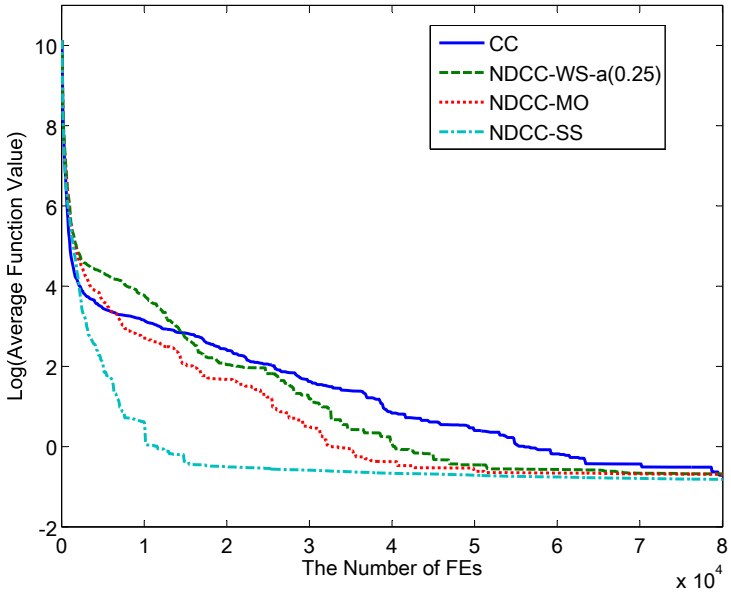


Figure 3.2: Evolution curves of CooC, NDCC-WS-a(0.25), NDCC-MO, and NDCC-SS on the single objective case ($a = 0.3$) in Scenario 1.

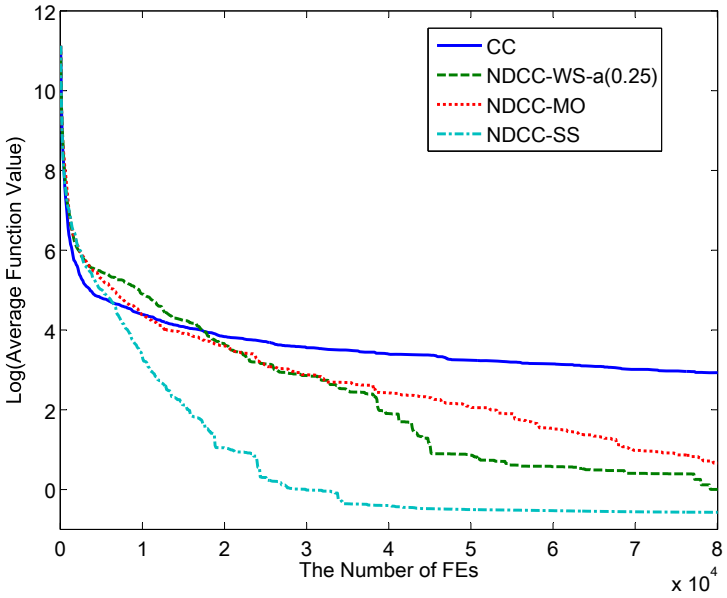


Figure 3.3: Evolution curves of CooC, NDCC-WS-a(0.25), NDCC-MO, and NDCC-SS on the single objective case ($a = 0.5$) in Scenario 1.

parameter p_n in the stochastic selection in NDCC-SS and conducted experiments on the single motor design problems. Three values for p_n were used, which are 0.25, 0.50, and 0.75. We also replaced the adaptive selection process in NDCC-SS with the multi-objective trade-off method and tested the performance of the resulting method on the single motor design problems. Table 3.4 summarises the statistical comparison results between NDCC-SS with these methods. The minus sign ($-$) in Table 3.4 denotes that the method in the corresponding row is statistically worse than NDCC-SS on the test problem in the corresponding column. From Table 3.4, we can observe that NDCC-SS performed better than any other method. This demonstrates the advantage of the use of the adaptive strategy in NDCC-SS.

Table 3.4: Comparison between NDCC-SS with NDCC-SS with $p_n = 0.25, 0.50, 0.75$ and multi-objective trade-off for single motor design problems. The best results are marked in **bold**.

Method	$a = 0.3$	$a = 0.5$
$p_n = 0.25$	5.13e−01±4.45e−02 −	6.14e−01±4.64e−02 −
$p_n = 0.50$	5.41e−01±4.68e−02 −	6.48e−01±5.29e−02 −
$p_n = 0.75$	1.54e+00±2.05e+00 −	7.73e+00±1.54e+01 −
Multi-objective	5.29e−01±4.14e−02 −	6.63e−01±1.41e−01 −
NDCC-SS	**4.43e−01±3.59e−02**	**5.66e−01±3.72e−02**

In the second experiment, we replaced the proposed novelty calculation in NDCC-SS by the novelty calculation proposed in [66] and tested the performance of the resulting method on single motor design problems. In the experiments, we considered two settings of archive in the novelty calculation, with and without an archive, and used four settings of p_0, p_f and r values in the adaptive stochastic selection. Table 3.5 summarises the statistical comparison results between NDCC-SS with these methods. The minus sign ($-$) in Table 3.5 denotes the method in the corresponding row is statistically worse than NDCC-SS on the test problem in the corresponding column. It can be seen from Table 3.5 that NDCC-SS outperformed the methods that used the existing novelty calculation. This demonstrates that the new novelty calculation strategy has better performance in addition to computational efficiency. This might be because that the new novelty calculation strategy can help maintain better diversity of the selected individuals than the existing novelty calculation strategy since the new novelty calculation is based on only the individuals that are already selected in the selection process of CooC while the existing one calculates individuals novelty based on the whole population before selection. As the selected individuals decide the search direction of CooC, the new novelty strategy may help preserve more diversity in the population and this is good for preventing CooC from converging to suboptimal states.

Table 3.5: Comparison between NDCC-SS with the methods that used the novelty calculation in [66]. The best results are marked in **bold**.

Method Archive/No(p_0,p_f,r)	$a = 0.3$	$a = 0.5$
No(0.5,0.25,0.4)	1.32e+00±4.58e+00 −	3.96e+00±8.72e+00 −
No(0.5,0.25,0.8)	5.42e−01±3.41e−01 −	4.98e+00±1.16e+01 −
No(0.25,0,0.4)	5.45e−01±7.14e−01 −	1.57e+01±3.18e+01 −
No(0.25,0,0.8)	1.36e+00±7.09e+00 −	1.34e+01±3.53e+01 −
Archive(0.5,0.25,0.4)	4.98e−01±2.22e−01 −	1.15e+00±3.01e+00 −
Archive(0.5,0.25,0.8)	4.93e−01±1.07e−01 −	1.16e+00±2.33e+00 −
Archive(0.25,0,0.4)	5.77e−01±6.02e−01 −	1.66e+01±3.06e+01 −
Archive(0.25,0,0.8)	4.86e−01±1.06e−01 −	7.58e+00±1.95e+01 −
NDCC-SS	**4.43e−01±3.59e−02**	**5.66e−01±3.72e−02**

In short, the additional comparative experimental results above showed the efficacy

of both the new novelty calculation and the adaptive stochastic selection in NDCC-SS. This is the reason why NDCC-SS performed well on single motor design problems, as shown in Table 3.3. Note that the new novelty calculation might not be the optimal novelty calculation method as it considers computational efficiency. More work is still worth doing along the direction of developing better novelty-driven CooC algorithms.

Scenario 2: Overlapping Motors Design

For this scenario, the proposed CCDM is compared to the baseline method that puts all design variables in a single optimisation process on the three overlapping motor design problems. For the baseline method, the population size was set to 100 in the experiments. For CCDM, the population size was set to 50. The maximum number of function evaluations was set to 32000 for the baseline method. While the baseline method needs to evaluate both f_1 and f_2 in a function evaluation on the overlapping motor design problems with two motors, CCDM only needs to evaluate one f_i $(i = 1, 2)$. Therefore, the maximum number of function evaluations was set to 32000*2 for CCDM. The parameter δ_f in CCDM was set to 0.005, and the ratio of the specified number of generations to the maximum number of generations was set to 0.8.

We also compare CCDM with the COSSOS method. For the COSSOS method, the population size for the supervisor-level optimisation was set to 20, and for each disciplinary-level optimisation, it was set to 50. As the COSSOS method also needs to evaluate only one f_i $(i = 1, 2)$ in a function evaluation in one disciplinary-level optimisation, the maximum number of function evaluations was set to 32000*2 for COSSOS. Besides, the maximum number of function evaluations for each disciplinary-level optimisation in COSSOS was set to 800 in our experiments. We independently ran each experiment for 100 times, and then used Wilcoxon rank-sum test at a significance level of 0.05 to conduct comparisons.

Table 3.6 presents the means and standard deviations of the best function values over 100 runs obtained by the three methods. The statistical comparison results between the

Table 3.6: Comparison between the baseline method, COSSOS and CCDM for overlapping motors design. The minus sign ($-$) and the approximation sign ($\approx$) denotes the compared method is statistically worse than, and similar to CCDM, respectively. The best results are marked in **bold**.

a values	Baseline Method	COSSOS method	CCDM
$a_1 = 0.10, a_2 = 0.125$	3.69e+00±1.61e+01 $-$	2.13e+00±6.63e+00 $-$	**6.87e−01±3.42e−01**
$a_1 = 0.10, a_2 = 0.30$	4.30e+00±1.66e+01 $-$	3.68e+00±6.81e+00 $-$	**2.47e+00±1.48e+01**
$a_1 = 0.05, a_2 = 0.50$	**1.57e+01±3.46e+01** $\approx$	1.51e+01±2.67e+01 $-$	**1.51e+01±1.02e+02**

two compared methods and CCDM are also given in this table. It can be seen from this table that both decomposition-based methods (i.e., CCDM and COSSOS) outperformed the baseline method. Between the two decomposition methods, the proposed CCDM achieved better solutions on all test problems.

The Geometric Programming Problem

For the geometric programming problem, the best solutions obtained by the all-at-once method (AAO) in [60] and the ATC with multi-objective formulation (ATC-MO) in [89] are: (z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, z_{10}, z_{11}, z_{12}, z_{13}, z_{14})=(2.84, 3.09, 2.36, 0.76, 0.87, 2.81, 0.94, 0.97, 0.87, 0.8, 1.3, 0.84, 1.76, 1.55) and (2.77, 3.14, 2.28, 0.76, 0.88, 2.86, 0.94, 0.96, 0.95, 0.85, 1.35, 0.84, 1.79, 1.58), respectively. Table 3.7 shows the corresponding f values and the constraint violation values of g and h constraints for these two solutions.

Table 3.7: Comparison between CCDM, AAO and ATC with the MO Formulation on the geometric programming problem. The best result is marked in **bod**.

	AAO [60]	ATC-MO [89]	CCDM
f	17.6137	17.5325	**17.3364**
g1	0.0002	0	0.0505
g2	0	0.0131	0.0473
g3	0.0078	0.0016	0.0393
g4	0.0128	0	0.0462
g5	0.0096	0.0356	0.0318
g6	0	0.0317	0.0193
h1	0.0078	0.0312	0
h2	0.0115	0.0220	0
h3	0.0550	0.0378	0
h4	0.0004	0.0490	0

It can be seen that both AAO and ATC-MO are unable to satisfy all constraints and the maximum violation tolerance for each constraint is 0.055 (see $h3$ in the column of AAO). In our experiments, CCDM was implemented with the same setting (0.055) for the maximum constraint violation tolerance. The population size for CCDM was set to 100, and the maximum number of function evaluations was set to 10000. The parameter δ_f was set to 0.01 and the ratio of the specified number of generations to the maximum number of generations was set to 0.3 for CCDM.

The best solution achieved by CCDM over 100 runs on this test problem was recorded. It is: (z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, z_{10}, z_{11}, z_{12}, z_{13}, z_{14})=(2.88, 3.00, 2.40, 0.73, 0.81, 2.77, 0.86, 0.99, 0.81, 0.76, 1.26, 0.97, 1.62, 1.59). Its f value and the constraint violation values of g and h constraints are given in Table 3.7. It can be seen that CCDM found a solution with better f value compared with AAO and ATC-MO.

3.3.4 The Effect of Communication Frequency

In this section, the best communication frequency among sub-populations in the proposed methods, with and without communication costs, is studied experimentally in the two scenarios of motor design problems and the geometric programming problem.

Best Communication Frequency without Communication Cost

In the first part, we study in both NDCC-SS and CCDM the best communication frequencies that obtain solutions of highest quality within a fixed number of function evaluations without considering communication costs. We ran NDCC-SS and CCDM on the test problems with each communication interval independently for 100 times and recorded the function values of the best solutions obtained using each communication interval over 100 runs.

As we wish to know which communication frequencies lead to significantly good performance, the Kruskal-Wallis test and a post-hoc test proposed by Conover and Iman

[24, 101] were applied with a significance level at 0.05. The Kruskal-Wallis test is a non-parametric multiple comparison test that compares more than two populations based on random samples by using rank sums. In our study, the Kruskal-Wallis test was applied on each test problem. The function value of the best solution achieved in one run was considered as a sample, and the 100 samples obtained using one communication frequency over 100 runs were considered as a population. The null hypothesis (H0) we used in the Kruskal-Wallis test states that all communication frequencies achieve the equivalent performance. If the null hypothesis is rejected, the communication frequency with the minimum rank sum will be selected as the control frequency, and the post-hoc test will be used to perform a comparison between the control frequency and every other communication frequency. To control the family-wise error rate (i.e., the probability of making one or more false discoveries when performing multiple hypotheses tests [34]), we further applied Finner's adjustment of p-values [39] in this multiple hypothesis testing. Any communication frequency whose performance is not significantly different from the control frequency will be recorded as the best communication frequency along with the control frequency.

The parameter settings of NDCC-SS and CCDM are the same as those in Section 3.3.3 except for the setting of communication frequencies. In this study, we considered 12 different communication intervals for the two scenarios of motor design problems and 12 different communication intervals for the geometric programming problem we used. They are $\{1, 2, 4, 5, 8, 10, 16, 20, 32, 40, 80, 160\}$ and $\{1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 80, 100\}$, respectively. Here, each number denotes the difference between the generation numbers of two sequential communications in the co-evolutionary process.

Table 3.8 gives the best communication intervals for NDCC-SS on the single motor design problems. It shows that it is best to exchange information among sub-populations every 1 or 2 generations when using NDCC-SS to solve the single motor design problems.

Table 3.9 presents the best communication intervals for CCDM on the overlapping motor design problems and the geometric programming problem. As it can be seen in Table 3.9, it is best to make sub-populations communicate with each other every generation

Table 3.8: Best communication intervals of NDCC-SS on scenario 1 without communication cost

a value	Best communication intervals
$a = 0.3$	1,2
$a = 0.5$	1,2

Table 3.9: Best communication intervals of CCDM on scenario 2 and the geometric programming problem without communication cost

Problem	Best communication intervals
$a_1 = 0.10, a_2 = 0.125$	1
$a_1 = 0.10, a_2 = 0.3$	1,2
$a_1 = 0.05, a_2 = 0.50$	1,2
Geometric programming	1,2,4,5,8,10,20,25,40,50,80,100

on the first design problem, every 1 or 2 generations on the second and third problems. For the geometric programming problem, all communication intervals performed comparatively well.

Our general conclusion based on Tables 3.8 and 3.9 is that sub-populations should communicate with each other as frequently as possible when applying the proposed methods to solve the test design problems without considering the communication cost.

Best Communication Frequency with Communication Cost

In the second part, we investigate the best communication frequency considering different communication costs. The communication cost is considered as a fixed number of function evaluations in our study. We varied the communication cost from $0 * NP$ to $40 * NP$ (NP denotes population size) function evaluations for all test problems, and carried out experiments using each communication interval from 1 to 20 generations for every communication cost. The other experimental settings are the same as in Section 3.3.3. We recorded the function values of the best solutions achieved by every communication interval over 100 runs. Then, the Kruskal-Wallis test with Conover's and Iman's post-hoc test as well as Finner's adjustment of p-values was used to give the best-performing communication frequency.

Tables 3.10 and 3.11 summarise the best communication intervals for NDCC-SS on the single motor design problems, and CCDM on the overlapping motor design problems and the geometric programming problem, respectively. A general observation from Tables 3.10 and 3.11 is that the best communication frequency varies as the design problem and the communication cost vary.

When comparing Tables 3.10-3.11 with Tables 3.8-3.9, we can see that the advantage of frequent communication was affected by the existence of communication cost. That is, one-generation communication is no longer always the best option when the communication cost is considered. This is because that more frequent communication will cause more time to be spent on communications rather than on optimisation. However, the situation is different for each problem. It can be seen from Table 3.10 that one generation is always among the best communication intervals for the design problem with $a = 0.5$ even when the communication cost is increased to $40 * NP$ function evaluations. The reason might be that the correlation between the decomposed sub-components needs the corresponding sub-populations to communicate with each other frequently even at a high cost.

Table 3.10: Best communication intervals of NDCC-SS on scenario 1 with different communication costs

Communication cost	$a = 0.3$	$a = 0.5$
$2 * NP$	2-8	1-4
$4 * NP$	1-20	1-8,10-11
$6 * NP$	1-20	1-20
$8 * NP$	1-20	1-20
$10 * NP$	3-20	1-20
$12 * NP$	1-20	1-20
$14 * NP$	2,5-20	1-20
$16 * NP$	4-20	1-20
$18 * NP$	1-20	1-20
$20 * NP$	3-20	1-20
$40 * NP$	1-20	1-20

Comparing each column in Tables 3.10-3.11, we can observe that larger communication intervals are more likely to be better when the communication cost becomes larger. Note that sometimes there is the trend that higher communication frequencies become better

Table 3.11: Best communication intervals of CCDM on scenario 2 and the geometric programming problem with different communication costs

Cost	$a_1 = 0.10$ $a_2 = 0.125$	$a_1 = 0.10$ $a_2 = 0.30$	$a_1 = 0.05$ $a_2 = 0.50$	Geometric programming
$2 * NP$	2-5	2-5	1-5	1-20
$4 * NP$	2-7	2-7	4-5	1-20
$6 * NP$	3-7	3-7	3-5	1-20
$8 * NP$	4-8	2-8	2,4-9	1-20
$10 * NP$	2-9,11	2-8	3-7,9	1-20
$12 * NP$	3-11	3-6	4-8,10	1-20
$14 * NP$	4-6, 9-10	2-7, 9-10	3-8, 12-13,20	1-20
$16 * NP$	3-9,18,20	5,7,8,19,20	3-11,16,18-20	1-20
$18 * NP$	8,17,20	3,5,20	3-5,7-10, 12,14-20	1-20
$20 * NP$	5-9,12,14, 16-17,20	3-8,10,12, 13, 15-20	4-7,12, 14,16-20	1-20
$40 * NP$	18-20	17-20	16-20	3-6, 8-20

again as communication cost becomes larger (e.g. one-generation communication is no longer the best at the cost of $16 * NP$ but becomes best again at the cost of $18 * NP$ in Table 3.10 where $a = 0.3$). As we use a fixed maximum number of function evaluations for different communication costs, the available number of function evaluations to do optimisation is different for different communication costs. Furthermore, the difference of the available function evaluations between different communication intervals becomes smaller as the communication cost increases. That means, the effect of communication cost on frequent communication becomes smaller as the communication cost increases. Consequently, frequent communication can become better again as the communication cost increases.

3.4 Communication Frequency Self-Adaptation

As the best communication frequencies for both proposed methods vary with the problem and communication cost, it is not a trivial task to find a good communication frequency for a new problem. Motivated by this, we propose a communication frequency self-adaptation

method (CFS) to automatically adjust a good communication frequency during the concurrent design process of NDCC-SS and CCDM.

3.4.1 The Proposed Self-Adaptive Method

The CFS method is inspired by the adaptation method proposed in [53], which is used to adapt the population size of an EA. The algorithm framework of the CFS method is given in Algorithm 7.

Algorithm 7 Communication Frequency Self-adaptation (CFS)

Randomly generate a communication interval: p_0
Generate p_1 and p_2 that satisfy $p_1 < p_0 < p_2$
while the stopping criterion is not met **do**
  Use each of $\{p_0,p_1,p_2\}$ for a specified number of function evaluations
  Calculate the performance improvement of each of $\{p_0,p_1,p_2\}$
  if p_1 achieves the largest improvement **then**
    set $p'_1 = \max(\lfloor p_1/2 \rfloor, p_L)$, $p'_0 = \max(p_1, p'_1 + 1)$, $p'_2 = \max(\lfloor (p_1 + p_0)/2 \rfloor, p'_0 + 1)$.
  else if p_2 achieves the largest improvement **then**
    set $p'_2 = \min(p_2 * 2, p_U)$, $p'_0 = \min(p_2, p'_2 - 1)$, $p'_1 = \min(\lceil (p_0 + p_2)/2 \rceil, p'_0 - 1)$.
  else if p_0 achieves the largest improvement **then**
    set $p'_0 = p_0$, $p'_1 = \min(\lceil (p_0 + p_1)/2 \rceil, p'_0 - 1)$, $p'_2 = \max(\lfloor (p_0 + p_2)/2 \rfloor, p'_0 + 1)$.
  else if all p_i (i=1,2,3) achieve the same performance improvement **then**
    set $p'_0 = p_0$, $p'_1 = \max(2 * p_1 - p_0, p_L)$, $p'_2 = \min(2 * p_2 - p_0, p_U)$.
  else if no improvement is achieved by any p_i (i=1,2,3) **then**
    set $p'_1 = \max(\lfloor p_1/4 \rfloor, p_L)$, $p'_0 = \max(\lfloor p_1/2 \rfloor, p'_1 + 1)$, $p'_2 = \max(\lfloor (p'_0 + p_1)/2 \rfloor, p'_0 + 1)$
  end if
  Set $p_0 = p'_0$, $p_1 = p'_1$, and $p'_2 = p_2$
end while

In the beginning, the CFS method randomly initialises a communication interval value p_0 from the pre-specified inteval range $[p_L, p_U]$. Then, two neighbors $\{p_1, p_2\}$ in different directions are generated for p_0. That is, $p_1 < p_0 < p_2$. The whole design process is divided into cycles in CFS. In each cycle, each of $\{p_0,p_1,p_2\}$ is used as the communication interval for a specified number of function evaluations. In this process, when p_i $(i = 0, 1, 2)$ is in turn, sub-populations in both NDCC-SS and CCDM communicate with each other every p_i generations, and then the obtained improvement is calculated and recorded. After this process, the value of p_0 is set to the p_i $(i = 0, 1, 2)$ that obtained the largest improvement.

Then, two new neighbors $\{p_1, p_2\}$ in different directions are generated for p_0 and the next cycle begins. The update of p_0 and the re-generation of two neighbors $\{p_1, p_2\}$ for the updated p_0 are based on the following greedy rules (for clarity, the new value of p_i $(i = 0, 1, 2)$ is denoted by p'_i $(i = 0, 1, 2)$):

- If p_1 achieves the largest improvement, then let $p'_1 = \max(\lfloor p_1/2 \rfloor, p_L)$, $p'_0 = \max(p_1, p'_1 + 1)$, and $p'_2 = \max(\lfloor (p_1 + p_0)/2 \rfloor, p'_0 + 1)$.

- If p_2 achieves the largest improvement, then let $p'_2 = \min(p_2 * 2, p_U)$, $p'_0 = \min(p_2, p'_2 - 1)$, and $p'_1 = \min(\lceil (p_0 + p_2)/2 \rceil, p'_0 - 1)$.

- If p_0 has the largest improvement, then let $p'_0 = p_0$, $p'_1 = \min(\lceil (p_0 + p_1)/2 \rceil, p'_0 - 1)$, and $p'_2 = \max(\lfloor (p_0 + p_2)/2 \rfloor, p'_0 + 1)$.

- If all p_i (i=0,1,2) achieve the same improvement, then let $p'_0 = p_0$, $p'_1 = \max(2 * p_1 - p_0, p_L)$, and $p'_2 = \min(2 * p_2 - p_0, p_U)$. This is to enlarge the search region.

- If no improvement is achieved by any p_i (i=0,1,2), then let $p'_1 = \max(\lfloor p_1/4 \rfloor, p_L)$, $p'_0 = \max(\lfloor p_1/2 \rfloor, p'_1 + 1)$, and $p'_2 = \max(\lfloor (p'_0 + p_1)/2 \rfloor, p'_0 + 1)$ to make sub-populations communicate more frequently to obtain improvement.

Note that each of p'_1, p'_0, and p'_2 needs to be kept as an integer and the relationship of $p'_1 < p'_0 < p'_2$ needs to be maintained during this process.

The degree of improvement of the performance is measured by the ratio of the improvement over either the best objective function value or constraint violation value. Furthermore, the improvement is normalised according to the efforts (the number of function evaluations) spent on it. For NDCC-SS on the single motor design problem, the improvement for each p_i $(i = 0, 1, 2)$ is defined as follows:

$$I = (f^t_{\text{best}} - f^{t+1}_{\text{best}})/(f^t_{\text{best}} * (Evals_{t+1} - Evals_t)) \tag{3.15}$$

where f^j_{best} $(j = t, t + 1)$ is the best objective function value at time j, and $Evals_j$ is the number of function evaluations spent until time j. For CCDM on the overlapping motor

design problems, if the best solution satisfies the constraints, the improvement calculation is the same as in Eq. (3.15). Otherwise, the improvement for each p_i $(i = 0, 1, 2)$ is calculated as the ratio of improvement on the constraint violation value normalised by the used number of function evaluations.

Note that it is not easy to determine how many function evaluations to run for each p_i in one cycle. First, we can not have a fixed value for all p_i values as it needs to be a multiple of all $(p_i * pop_{\text{num}} * NP + cost)$ where pop_{num} is the number of sub-populations, NP is the size of sub-populations, and $cost$ is the communication cost. Otherwise, a fixed p_i value would be too large and thus, the number of cycles would be too small to do an effective self-adaptation. Second, if we can not have a fixed number of function evaluations for all p_i, this method would have some biases. For example, if p_0 and p_1 always get less function evaluations than p_2 does in every cycle, the method might perform badly on problems that prefer small communication intervals. To minimise the bias, we run p_2 for $(p_2 * pop_{\text{num}} * NP + cost)$ function evaluations, and run each p_i $(i = 0, 1)$ for a randomly selected number of function evaluations between $\lfloor (p_2 * pop_{\text{num}} * NP + cost)/(p_i * pop_{\text{num}} * NP + cost) \rfloor * (p_i * pop_{\text{num}} * NP + cost)$ and $\lceil (p_2 * pop_{\text{num}} * NP + cost)/(p_i * pop_{\text{num}} * NP + cost) \rceil * (p_i * pop_{\text{num}} * NP + cost)$ in each cycle.

3.4.2 Experimental Results

In this section, we assume a fixed communication cost and study the performance of the proposed CFS method by comparing it with a random method and a method of constant communication frequency in both NDCC-SS and CCDM. In the random method, at each communication step, it randomly decides the next communication time from the specified communication interval range. In fixed communication method, a fixed communication interval of 1 generation is used.

In the experiments, we set the communication cost to $pop_{num} * NP$ and the communication interval range to [1, 50]. The maximum number of function evaluations ($MaxFEs$) was set to $2 * 10^5$, and the specified number of generations in both NDCC-SS and CCDM

was set to $\lfloor MaxFEs/(pop_{\text{num}} * NP + cost) \rfloor$. Other settings are the same as in Section 3.3.3.

Tables 3.12 and 3.13 summarise the means and standard deviations of the function values of the best solutions obtained over 100 runs by using each of the random method, the fixed method and the adaptive method in NDCC-SS and CCDM. We used the Wilcoxon rank-sum test at a significance level of 0.05 to make comparisons between the CFS method and the two compared methods over each test problem, and the statistical test results are also given in Tables 3.12 and 3.13. The signs '≈' and '−' in Tables 3.12 and 3.13 denote the compared method is similar to and significantly worse than the proposed CFS method, respectively.

According to the statistical test results, the CFS method performed similarly to the random method on the geometric programming problem in CCDM, and performed better than it on the single motor design problems in NDCC-SS and on the overlapping motors design problems in CCDM.

When compared to the fixed method, the CFS method performed better on one test problem and similarly on the other test problem in NDCC-SS. When applied in CCDM, the CFS method and the fixed method obtained similar results on the geometric programming problem. However, the CFS method performed better than the fixed method on the other test problems.

Table 3.12: Comparison between the CFS method and the compared methods for NDCC-SS on scenario 1. The best results are marked in **bold**.

Problem	Random method	Fixed method	CFS method
$a = 0.3$	1.34e+00±5.71e+00 −	4.87e−01±4.13e−02 −	**4.54e−01±4.77e−02**
$a = 0.5$	1.65e+01±3.62e+01 −	**5.80e−01±3.94e−02** ≈	**2.97e+00±1.33e+01**

Table 3.13: Comparison between the CFS method and the compared methods for CCDM on scenario 2 and the geometric programming problem. The best results are marked in **bold**.

Problem	Random method	Fixed method	CFS method
$a_1 = 0.10, a_2 = 0.125$	1.47e+00±3.83e+00 −	1.02e+00±2.04e+00 −	**1.01e+00±1.89e+00**
$a_1 = 0.10, a_2 = 0.3$	4.09e+00±1.42e+01 −	1.77e+01±1.17e+02 −	**9.99e−01±2.38e+00**
$a_1 = 0.05, a_2 = 0.5$	3.10e+01±2.59e+02 −	1.26e+01±6.42e+01 −	**2.36e+00±1.33e+01**
Geometric programming	**4.20e+01±2.43e+01** ≈	**4.97e+01±2.25e+01** ≈	**4.78e+01±2.54e+01**

According to these comparison results, we can see clearly that the CFS method is an effective method for both NDCC-SS and CCDM when information about how to set the communication frequency is unavailable beforehand. The results also show that all of the random method, the fixed method, and the CFS method performed similarly on the geometric programming problem. This is because the use of different communication intervals led to similar results on this problem. The CFS method can not bring any performance improvement on such problems.

3.5 Chapter Summary

This chapter developed efficient CooC-based design optimisation methods to address two types of CE problems and to answer questions of *how to better evaluate individuals in each sub-population when using CooC to address the first type of CE problem*, *how to better handle the shared design variables when using CooC to address the second type of CE problem*, and *how often different sub-populations should communicate with each other in the CooC-based concurrent design optimisation* .

As an answer to the first question, this chapter proposed a new novelty-driven CooC method, NDCC-SS, to address the first type of CE problems. Novelty-driven CooC evaluates individuals based not only on their fitness but also on their novelty. In NDCC-SS, a computationally efficient novelty calculation was used and a stochastic selection process with an adaptive probability was used to adjust a good trade-off between novelty and fitness when making comparisons between individuals. The experimental results on the UEM design problems have shown the new novelty calculation strategy along with the adaptive trade-off strategy provides a good choice to evaluate individuals and compare them.

As an answer to the second question, this chapter proposed CCDM method which used duplicate variables and the stochastic ranking method to deal with the common variables for quasi-separable MDO problems. The superior performance of CCDM has

been demonstrated through the experimental comparison with other MDO methods on the UEM design problems and a general MDO problem.

As an answer to the third question, this chapter studied the performance effect of the communication frequency on NDCC-SS and CCDM. By using different problems and communication costs, the experimental results showed that the best frequency varies as the problem and communication cost change. Motivated by this, a self-adaptive method, CFS, was developed to adjust a good communication frequency automatically during the concurrent design process. The experimental studies showed that CFS is a very competitive communication strategy for both NDCC-SS and CCDM.

CHAPTER 4

A NEW SPECIATION METHOD FOR DYNAMIC NT NED OPTIMISATION

The CE problems considered in Chapter 3 are static. In the field of CE, changes in the design objectives or constraints usually happen suddenly due to the varying customer needs, and fast response to the changes is required [110, 130]. Besides, such dynamic constrained optimisation problems (DCOPs) also widely exist in other fields due to the dynamic environments, such as vehicle routing [44], load balancing [134], service composition [57], etc. This chapter focuses on how to address DCOPs efficiently.

In Chapter 3, we assume how to decompose the two types of CE problems are known beforehand, and based on this we studied how to better co-evolve the decomposed subproblems. However, in many real-world applications, we may not have any prior knowledge about how to decompose the problem, e.g., how to decompose a DCOP. This chapter aims to address this issue on DCOPs through the use of niching EAs. Before giving the details about the proposed method in Section 4.2, the motivation of this work will be first elaborated in Section 4.1. In Section 4.3, experimental studies to validate the efficacy of the proposed method will be presented. Finally, Section 4.4 will summarise this chapter and give conclusions.

4.1 Motivation

In the real world, many optimisation problems are changing over time due to dynamic environments [25], [92]. These problems require an optimisation algorithm to quickly find the new optimum once the problem changes [93]. As a class of nature-inspired optimisation methods, evolutionary algorithms (EAs) promise a potentially good adaptation capability to changing environments, and thus have been widely studied in the field of dynamic optimisation (DO). Many evolutionary DO approaches have been developed, which include maintaining/introducing diversity strategies, memory approaches, prediction approaches, multi-population approaches, and so on [93].

As revealed in [94, 93], most existing studies of DO focus on unconstrained DOPs or DCOPs with box constraints, and few consider DCOPs that involve linear or nonlinear constraints despite their high popularity in real-world applications. In a DCOP, a change may occur in either constraints or objective functions or both. Therefore, DCOPs have some specific characteristics compared to unconstrained or bound-constrained DOPs. For a DCOP, the distribution of infeasible/feasible solutions might change, and the global optima might move from one feasible region to another disconnected feasible region, or appear in a new region without changing the current optima due to the dynamics of environments [94, 92].

The specific characteristics of DCOPs pose difficulties for some existing DO strategies and constraint handling (CH) techniques in addressing them. As discussed in [92] and [94], maintaining/introducing diversity methods such as random-immigrants (RI) [50] and hyper-mutation (HyperM) [22] become less effective on DCOPs than on DOPs, when combined with a penalty function that prefers feasible solutions to infeasible ones. The reasons are as follows. Although random individuals are generated and introduced into the current population after a change happens, as most of them are likely infeasible, they will be abandoned by the penalty function which prefers feasible solutions. As a result, the diversity of population can not be maintained or increased. However, without enough population diversity, the RI and HyperM method can not deal well with the situation

in which the global optimum moves from one feasible region to another disconnected feasible region, as this situation needs an infeasible path from the previous optimum to the current optimum. Moreover, some adaptive/self-adaptive CH techniques also face challenges in solving DCOPs as they need the knowledge of problem that is unavailable in a dynamic environment or historical information that might be outdated once the problem changes [94].

To address these difficulties, the authors in [94] suggested using constraint handling techniques that can accept diversified infeasible solutions or tracking moving feasible regions when dealing with DCOPs. Researchers have also carried out some work following this suggestion. As introduced in Section 2.6.2 in Chapter 2, to allow diversified infeasible solutions distributed in the whole search space, the authors in [92, 97, 5] applied the repair method [112] to handle constraints along with RI/HyperM to deal with changes. The repair method repairs every infeasible solution to a feasible one, and uses the fitness of the feasible solution to evaluate it. Consequently, infeasible solutions that can make a good feasible solution will be kept in the population and thus the population diversity can be maintained. However, these methods need a lot of feasibility checkings, and thus cannot be applied to DCOPs in which the ratio of feasible solutions is very low and a feasibility checking is computationally costly.

A simple ranking scheme in [54] was applied to handling constraints in [15]. To maintain population diversity, the method monitored the population diversity and switched between a global search and a local search operator according to whether the diversity degree is larger than a threshold. As a result, this method will highly depend on the setting of the threshold. To avoid the setting of the threshold, the authors in [16] used the Shannon's index of diversity as a factor to balance the influence of the global-best and local-best search directions. However, the population tends to converge in this method, and it might be ineffective to make the partially converged population to re-diversify to track the switched global optimum.

The authors in [4] employed simple feasibility rules [85] as the CH strategy along

with RI and combined differential evolution (DE) variants to introduce diversity after each change. However, this method may not maintain diversity well during the run as infeasible solutions are still likely abandoned by feasibility rules. Except DO strategies to introduce/maintain diversity were considered, other DO strategies such as memory and prediction methods were also combined with CH techniques to deal with DCOPs. The study in [107] adapted abstract memory method, and the infeasibility driven evolutionary algorithm (IDEA) was combined with prediction method (to predict the future optima) in [37, 38] to handle DCOPs. However, both memory and prediction methods are only applicable to particular dynamic problems (i.e., cyclic and predictable dynamic problems, respectively).

In the aforementioned methods, none of them were done along the direction of tracking moving feasible regions. However, in the field of unconstrained DOPs or DCOPs with bounded constraints, using niching EAs to locate and track moving optima has become one of the most important techniques [69, 12, 86, 100]. Motivated by the capability of niching EAs in solving unconstrained DOPs or DCOPs with bounded constraints, this chapter aims to apply niching methods to address DCOPs and study the question of *whether niching method can help better solve DCOPs*. As introduced in Section 2.3.1 in Chapter 2, niching EAs can locate multiple optima by explicitly or implicitly maintaining multiple sub-populations to search different promising regions of the decision space. In this chapter, niching EAs are employed to locate and track multiple feasible regions. Research questions of *how to locate multiple feasible regions in a DCOP* and *how to track moving feasible regions in a DCOP* are studied. As niching methods focus on exploration, newly changed optima might not be found quickly as promising regions are not exploited sufficiently. Therefore, the question of *how to make a good trade-off between exploration and exploitation* is also investigated.

To answer these questions, this chapter introduced a new speciation-based method, called speciated evolution with local search (SELS), to address the challenges of DCOPs. To locate multiple feasible regions, SELS employs deterministic crowding (DC) [80] and

assortative mating (AM) [28] to induce speciation in the population. DC and AM make recombination and comparisons between similar individuals, respectively. Moreover, SELS uses the simple and parameter-free feasibility rules [85] which prefer feasible solutions to infeasible solutions to deal with constraints. Consequently, good solutions can be maintained in different feasible regions. To track moving feasible regions, SELS adds random immigrants into the population to introduce diversity once a change is detected. Furthermore, to make a good trade-off between exploration and exploitation, a local search strategy is employed in SELS to promote exploitation of the promising regions to quickly find the changed optimum. Finally, the performance of SELS is demonstrated on 11 DCOP benchmark test functions.

4.2 The Proposed SELS Method

Algorithm 8 presents the framework of SELS. It begins with a randomly generated population of candidate solutions and utilises genetic algorithm (GA) to evolve this population. At every generation, some individuals in the population will be re-evaluated to detect the changes. If the change happens, the whole population will be re-evaluated and some random immigrants will be introduced. Otherwise, the whole population will be evolved using DC, AM and local search operations. The parameter ls_{num} in Algorithm 8 denotes the maximum number of fitness evaluations permitted for each local search. Note that NP needs to be set to an even number.

4.2.1 Deterministic Crowding (DC) and Assortative Mating (AM)

In this work, the DC method [80] is chosen among existing niching techniques. Details about DC can be found in Algorithm 2 in Section 2.3.1 in Chapter 2. The DC method pairs all population elements randomly and generates two offspring for each pair based on EA operators. Selection is then operated on each group of two parents and two offspring, and a similarity measure is used to decide which offspring competes against which parent.

Algorithm 8 The Framework of SELS

```
1: Evaluate a randomly generated population P = {x_i|i = 1, 2, ..., NP}, NP is even.
2: while computational resources are not used up do
3:    repeat
4:       Randomly select an unpaired individual x from the population
5:       Calculate Euclidean distance between x and each other unpaired individual
6:       Pair x with the individual that has smallest positive Euclidean distance to x
7:    until all individuals in the population are paired
8:    for i ← 1, 2, ..., NP do
9:       if i or i + 1 is one detection index then
10:         Re-evaluate x_i or x_{i+1}
11:      end if
12:      if the change is detected then
13:         Re-evaluate solutions in P and introduce diversity, go to Step 20
14:      else
15:         Generate two offspring o_1, o_2 from x_i and x_{i+1} with crossover and mutation
16:         Do deterministic crowding(x_i, x_{i+1}, o_1, o_2) according to Algorithm 2 in Section
            2.3.1 in Chapter 2
17:         Set i = i + 2
18:      end if
19:   end for
20:   Do local search on the best solution in P for ls_num fitness evaluations
21: end while
```

The offspring will replace the compared parent and enter next generation if it is fitter. Compared to other niching techniques, DC does not need any parameter setting. This is why DC is chosen in SELS.

As DC can maintain good solutions on different peaks or in different feasible regions, intuitively, we would not like to conduct crossover between solutions on different peaks or in different feasible regions as doing this will likely generate solutions in the valley or infeasible regions. To avoid this, the AM method [28] is used, which mates individuals with the most similar non-identical partner in the population. Through doing crossover between individuals in proximity, species will be automatically generated, and the exploitation of the corresponding search area will also be enhanced.

In both DC and AM, the similarity between two individuals, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, is measured by the Euclidean distance between them. The smaller the distance between them is, the more similar they are. Eq. (4.1) shows the calculation process of the Euclidean distance

between x_i and x_j:

$$Dist(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{\sum_{k=1}^{n}(x_{i,k} - x_{j,k})^2} \tag{4.1}$$

where n represents the dimension of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.

4.2.2 Feasibility Rules

To deal with constraints and make comparisons between individuals, one of the most popular constraint handling techniques, feasibility rules [85], is used in SELS to determine the fitter one in each pair of parent and offspring in the deterministic crowding. Compared to other constraint handling techniques, the method of feasibility rules is very simple and parameter-free. It makes comparison between two individuals according to the following three selection criteria:

1. If both of the individuals are feasible solutions, the one with the higher fitness value is selected.

2. If both of the individuals are infeasible solutions, the one with the lower sum of constraint violation wins.

3. If one of the individuals is feasible and the other is infeasible, the feasible individual is preferred.

4.2.3 Local Search (LS) Strategy

In this work, the local search framework, local evolutionary search enhancement by random memorizing (LESRM) [125], is applied to the best solution (x_{best}) in the current population at each generation. LESRM applies an EA that adaptively adjusts its mutation step size to evolve the current individual and adjusts the search direction based on individuals encountered before. Thus, it can do efficient exploitation in the area that the best solution is located in. In this work, we instantiate LESRM with a (1+1)-

evolutionary strategy (ES). Algorithm 9 gives the pseudo-code for LESRM which uses a memory *Memory* to archive the previous best solutions.

Algorithm 9 Evolutionary Search in LESRM in SELS ($\boldsymbol{x}_{best}$)

1: Set $Memory = \emptyset$
2: Set the beam search factor $b = 2$
3: Find the closest non-identical solution $\boldsymbol{x}_{near}$ in the current population to $\boldsymbol{x}_{best}$
4: Set $\delta = Dist(\boldsymbol{x}_{near}, \boldsymbol{x}_{best})$
5: Set local search generation counter $ls_g = 0$
6: **repeat**
7: **repeat**
8: Set $ls_g = ls_g + 1$
9: Generate a offspring $\boldsymbol{x}_{offspring} = \boldsymbol{x}_{best} + \delta$*randn(1,D);
10: **if** mod(ls_g, $period$)==0 **then**
11: Calculate the success ratio of δ
12: Set $\delta = \delta * 2$ if the success ratio of δ is larger than 0.5
13: Set $\delta = \delta/2$ if the success ratio of δ is less than 0.5
14: **end if**
15: **until** $\boldsymbol{x}_{offspring}$ is fitter than $\boldsymbol{x}_{best}$
16: Update the *Memory* with $\boldsymbol{x}_{best}$
17: Set $\boldsymbol{x}_{best} = \boldsymbol{x}_{offspring}$
18: **if** *Memory* is not empty **then**
19: Randomly draw a solution $\boldsymbol{m}_r$ from the *Memory*
20: Calculate the direction s from $\boldsymbol{m}_r$ to $\boldsymbol{x}_{best}$, $s = (\boldsymbol{x}_{best} - m_r)/||\boldsymbol{x}_{best} - m_r||$
21: Set $b_a = 1$
22: **repeat**
23: $b_a = b_a * b$
24: $\boldsymbol{x}_{new} = \boldsymbol{x}_{best} + s * b_a * \delta$
25: **if** $\boldsymbol{x}_{new}$ is fitter than $\boldsymbol{x}_{best}$ **then**
26: Update the *Memory* with $\boldsymbol{x}_{best}$
27: Set $\boldsymbol{x}_{best} = \boldsymbol{x}_{new}$
28: **end if**
29: **until** $\boldsymbol{x}_{best}$ is fitter than $\boldsymbol{x}_{new}$
30: **end if**
31: Update the *Memory* with $\boldsymbol{x}_{new}$
32: **until** ls_{num} function evaluations are used up

At each iteration in the local search, LESRM first uses ES to evolve the current individual until a better individual is generated (see Steps 7-15). During the evolutionary process of ES, the mutation step size δ is adjusted based on its success ratio. The success ratio of δ is calculated as the ratio of the offspring generated using δ that are better than the current individual among all the offspring generated using δ in a period. If the

success ratio is bigger than 0.5, then δ is doubled to extend the search range (see Step 12). Otherwise, the δ is set to $\delta/2$ to narrow the search range (see Step 13). In this work, *period* is set to 2.

Then, the current individual is replaced by the better generated offspring (see Step 17). After this, LESRM randomly selects one individual from the memory *Memory*, and calculates the direction from this randomly selected individual to the current individual (see Steps 19-20). The calculated direction is then used to generate new individuals until the newly generated individual is worse than the current individual (see Steps 22-29). In this process, if the newly generated individual is better than the current individual, the current individual will be replaced by the new individual (see Steps 25-27).

The above process is iterated until ls_{num} function evaluations are used up. The memory depth is set to $d_m = 2 * n * n^2 * n^3$ (n denotes the dimension) in [125]. Once the memory arrives at the maximum depth, the oldest individuals will be first replaced by the newly added individuals.

4.2.4 Change Detection and Diversity Introduction

To detect the change in time, at every generation, SELS re-evaluates the (NP/k)-th, $(2 * NP/k)$-th, $((k-1) * NP/k)$-th, ..., (NP)-th individual in the population. Here, k is the number of individuals for detection and NP is the size of population. Once the fitness or the constraint violation values of the detection individual changes, the whole population will be re-evaluated. After this, to introduce diversity into the population, NI individuals are first randomly generated. They are then used to randomly replace some individuals in the population except the best individual to do partial re-initialisation. In this work, NI is set according to the severity of change as follows. Here, for each pair of individuals $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in the population, Compare$(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is used to denote whether $\boldsymbol{x}_i$ is better than $\boldsymbol{x}_j$. If $\boldsymbol{x}_i$ is better than or the same to $\boldsymbol{x}_j$ according to the feasibility rules, Compare$(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is equal to 1. Otherwise, Compare$(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is equal to 0. After the re-evaluation, if Compare$(\boldsymbol{x}_i, \boldsymbol{x}_j)$ changes, then this is recorded as one reverse order.

The degree of change severity after re-evaluation is estimated as the ratio of the number of reverse order over all pairs of individuals in the population. Finally, NI is set to $\max(\lceil reverse\_ratio * NP \rceil, 2)$.

4.3 Experimental Studies

4.3.1 Experimental Setup

To assess the efficacy of SELS, we conducted experiments on 11 DCOP benchmark test functions proposed in [94]. There are actually 18 test functions in total in the benchmark. However, only 11 of them satisfy the definition of DCOPs given in Eq. (2.3). The 11 test functions are G24-l(dF,fC), G24-2(dF,fC), G24-3(dF,dC), G24-3b(dF,dC), G24-4(dF,dC), G24-5(dF,dC), G24-6a(dF,fC), G24-6c(dF,fC), G24-7(fF,dC), G24-6d(dF,fC), and G24-8b(fC,dF), respectively. Here, 'dF', 'fF', 'dC' and 'fC' mean dynamic objective function, fixed objective function, dynamic constraint functions, and fixed constraint functions, respectively.

All of the 11 test functions have two decision variables: x_1 and x_2. The range of them are $[0, 3]$ and $[0, 4]$, respectively. They were developed based on two basic objective functions and six constraint functions. The two basic objective functions are:

$$\begin{aligned} f_1 &= -(X_1 + X_2) \\ f_2 &= -3\exp(-\sqrt{\sqrt{X_1^2 + X_2^2}}) \end{aligned} \tag{4.2}$$

and the six constraint functions are:

$$
\begin{aligned}
g_1 &= -2Y_1^4 + 8Y_1^3 - 8Y_1^2 + Y_2 - 2 \\
g_2 &= -4Y_1^4 + 32Y_1^3 - 88Y_1^2 + 96Y_1 + Y_2 - 36 \\
g_3 &= 2Y_1 + 3Y_2 - 9 \\
g_4 &= \begin{cases} -1 & \text{if } (0 \le Y_1 \le 1) \text{ or } (2 \le Y_1 \le 3) \\ 1 & \text{otherwise} \end{cases} \\
g_5 &= \begin{cases} -1 & \text{if } (0 \le Y_1 \le 0.5) \text{ or } (2 \le Y_1 \le 2.5) \\ 1 & \text{otherwise} \end{cases} \\
g_6 &= \begin{cases} -1 & \text{if } [(0 \le Y_1 \le 1) \text{ or } (2 \le Y_2 \le 3)] \text{ or } (2 \le Y_1 \le 3) \\ 1 & \text{otherwise} \end{cases}
\end{aligned}
\tag{4.3}
$$

where X_i $(i = 1, 2)$ and Y_i $(i = 1, 2)$ satisfy:

$$
\begin{aligned}
X_i(x,t) &= p_i(t)(x_i + q_i(t)) \\
Y_i(x,t) &= r_i(t)(x_i + s_i(t))
\end{aligned}
\tag{4.4}
$$

The parameters $p_i(t), q_i(t), r_i(t), s_i(t)$ are the environmental parameters that changes as the time t goes by.

The definitions of the 11 test functions based on the two basic objective functions and six constraint functions are as follows:

1. G24-l(dF,fC):

$$
\begin{aligned}
&\min_{\boldsymbol{x}} \ f_1(\boldsymbol{x}) \\
&\text{subject to} : g_1(\boldsymbol{x}) \le 0, \ g_2(\boldsymbol{x}) \le 0
\end{aligned}
\tag{4.5}
$$

The dynamic changes of $p_i(t), q_i(t), r_i(t), s_i(t)$ in G24-l(dF,fC) are:

$$\begin{aligned} &p_1(t) = sin(k\pi t + \pi/2),\ p_2(t) = 1,\ q_i(t) = 0,\ i = 1, 2 \\ &r_i(t) = 1,\ s_i(t) = 0,\ i = 1, 2 \end{aligned} \tag{4.6}$$

where k denotes the change severity. Different change severity can be considered by setting k to different values.

2. G24-2(dF,fC):

$$\begin{aligned} &\min_{\boldsymbol{x}}\ f_1(\boldsymbol{x}) \\ &\text{subject to}: g_1(\boldsymbol{x}) \leq 0,\ g_2(\boldsymbol{x}) \leq 0 \end{aligned} \tag{4.7}$$

The dynamic changes of $p_i(t), q_i(t), r_i(t), s_i(t)$ in G24-2(dF,fC) are:

$$\begin{aligned} &\text{if } (t \bmod 2 = 0) \begin{cases} p_1(t) = \sin(k\pi t/2 + \pi/2) \\ p_2(t) = \begin{cases} p_2(t-1), & \text{if } t > 0 \\ p_2(0) = 0, & \text{if } t = 0 \end{cases} \end{cases} \\ &\text{if } (t \bmod 2 \neq 0) \begin{cases} p_1(t) = \sin(k\pi t/2 + \pi/2) \\ p_2(t) = \sin(k\pi(t-1)/2 + \pi/2) \end{cases} \\ &q_i(t) = 0,\ r_i(t) = 1,\ s_i(t) = 0,\ i = 1, 2 \end{aligned} \tag{4.8}$$

3. G24-3(dF,dC):

$$\begin{aligned} &\min_{\boldsymbol{x}}\ f_1(\boldsymbol{x}) \\ &\text{subject to}: g_1(\boldsymbol{x}) \leq 0,\ g_2(\boldsymbol{x}) \leq 0 \end{aligned} \tag{4.9}$$

The dynamic changes of $p_i(t), q_i(t), r_i(t), s_i(t)$ in G24-3(dF,dC) are:

$$\begin{aligned} &p_i(t) = 1,\ q_i(t) = 0,\ i = 1, 2 \\ &r_i(t) = 1,\ s_1(t) = 0,\ s_2(t) = 2 - t * (x_2^{max} - x_2^{min})/S,\ i = 1, 2 \end{aligned} \tag{4.10}$$

where S denotes the change severity. Different change severity can be considered by setting S to different values. x_2^{max} and x_2^{min} denote the upper and lower bounds of x_2, respectively.

4. G24-3b(dF,dC):

$$\begin{aligned} &\min_{\boldsymbol{x}}\ f_1(\boldsymbol{x}) \\ &\text{subject to}: g_1(\boldsymbol{x}) \leq 0,\ g_2(\boldsymbol{x}) \leq 0 \end{aligned} \tag{4.11}$$

The dynamic changes of $p_i(t), q_i(t), r_i(t), s_i(t)$ in G24-3b(dF,dC) are:

$$\begin{aligned} &p_1(t) = sin(k\pi t + \pi/2),\ p_2(t) = 1,\ q_i(t) = 0,\ i = 1, 2 \\ &r_i(t) = 1,\ s_1(t) = 0,\ s_2(t) = 2 - t * (x_2^{max} - x_2^{min})/S,\ i = 1, 2 \end{aligned} \tag{4.12}$$

5. G24-4(dF,dC):

$$\begin{aligned} &\min_{\boldsymbol{x}}\ f_1(\boldsymbol{x}) \\ &\text{subject to}: g_1(\boldsymbol{x}) \leq 0,\ g_2(\boldsymbol{x}) \leq 0 \end{aligned} \tag{4.13}$$

The dynamic changes of $p_i(t), q_i(t), r_i(t), s_i(t)$ in G24-4(dF,dC) are:

$$\begin{aligned} &p_1(t) = sin(k\pi t + \pi/2),\ p_2(t) = 1,\ q_i(t) = 0,\ i = 1, 2 \\ &r_i(t) = 1,\ s_1(t) = 0,\ s_2(t) = t * (x_2^{max} - x_2^{min})/S,\ i = 1, 2 \end{aligned} \tag{4.14}$$

6. G24-5(dF,dC):

$$\begin{aligned} &\min_{\boldsymbol{x}} \ f_1(\boldsymbol{x}) \\ &\text{subject to}: g_1(\boldsymbol{x}) \leq 0, \ g_2(\boldsymbol{x}) \leq 0 \end{aligned} \tag{4.15}$$

The dynamic changes of $p_i(t), q_i(t), r_i(t), s_i(t)$ in G24-5(dF,dC) are:

$$\begin{aligned} &\text{if } (t \bmod 2 = 0) \begin{cases} p_1(t) = \sin(k\pi t/2 + \pi/2) \\ p_2(t) = \begin{cases} p_2(t-1), & \text{if } t > 0 \\ p_2(0) = 0, & \text{if } t = 0 \end{cases} \end{cases} \\ &\text{if } (t \bmod 2 \neq 0) \begin{cases} p_1(t) = \sin(k\pi t/2 + \pi/2) \\ p_2(t) = \sin(k\pi(t-1)/2 + \pi/2) \end{cases} \\ &q_i(t) = 0, \ r_i(t) = 1, \ s_1(t) = 0, \ s_2(t) = t * (x_2^{max} - x_2^{min})/S, \ i = 1, 2 \end{aligned} \tag{4.16}$$

7. G24-6a(dF,fC):

$$\begin{aligned} &\min_{\boldsymbol{x}} \ f_1(\boldsymbol{x}) \\ &\text{subject to}: g_3(\boldsymbol{x}) \leq 0, \ g_6(\boldsymbol{x}) \leq 0 \end{aligned} \tag{4.17}$$

The dynamic changes of $p_i(t), q_i(t), r_i(t), s_i(t)$ in G24-6a(dF,fC) are:

$$\begin{aligned} &p_1(t) = sin(k\pi t + \pi/2), \ p_2(t) = 1, \ q_i(t) = 0, \ i = 1, 2 \\ &r_i(t) = 1, \ s_i(t) = 0, \ i = 1, 2 \end{aligned} \tag{4.18}$$

8. G24-6c(dF,fC):

$$\begin{aligned} &\min_{\boldsymbol{x}} \ f_1(\boldsymbol{x}) \\ &\text{subject to}: g_3(\boldsymbol{x}) \leq 0, \ g_4(\boldsymbol{x}) \leq 0 \end{aligned} \tag{4.19}$$

and the dynamic changes of $p_i(t), q_i(t), r_i(t), s_i(t)$ in G24-6c(dF,fC) are:

$$\begin{aligned} &p_1(t) = sin(k\pi t + \pi/2),\ p_2(t) = 1,\ q_i(t) = 0,\ i = 1, 2 \\ &r_i(t) = 1,\ s_i(t) = 0,\ i = 1, 2 \end{aligned} \tag{4.20}$$

9. G24-6d(dF,fC):

$$\begin{aligned} &\min_{\boldsymbol{x}}\ f_1(\boldsymbol{x}) \\ &\text{subject to}: g_5(\boldsymbol{x}) \leq 0,\ g_6(\boldsymbol{x}) \leq 0 \end{aligned} \tag{4.21}$$

The dynamic changes of $p_i(t), q_i(t), r_i(t), s_i(t)$ in G24-6d(dF,fC) are:

$$\begin{aligned} &p_1(t) = sin(k\pi t + \pi/2),\ p_2(t) = 1,\ q_i(t) = 0,\ i = 1, 2 \\ &r_i(t) = 1,\ s_i(t) = 0,\ i = 1, 2 \end{aligned} \tag{4.22}$$

10. G24-7(fF,dC):

$$\begin{aligned} &\min_{\boldsymbol{x}}\ f_1(\boldsymbol{x}) \\ &\text{subject to}: g_1(\boldsymbol{x}) \leq 0,\ g_2(\boldsymbol{x}) \leq 0 \end{aligned} \tag{4.23}$$

The dynamic changes of $p_i(t), q_i(t), r_i(t), s_i(t)$ in G24-7(fF,dC) are:

$$\begin{aligned} &p_i(t) = 1,\ q_i(t) = 0,\ i = 1, 2 \\ &r_i(t) = 1,\ s_1(t) = 0,\ s_2(t) = t * (x_2^{max} - x_2^{min})/S,\ i = 1, 2 \end{aligned} \tag{4.24}$$

11. G24-8b(fC,dC):

$$\begin{aligned} &\min_{\boldsymbol{x}}\ f_2(\boldsymbol{x}) \\ &\text{subject to}: g_1(\boldsymbol{x}) \leq 0,\ g_2(\boldsymbol{x}) \leq 0 \end{aligned} \tag{4.25}$$

The dynamic changes of $p_i(t), q_i(t), r_i(t), s_i(t)$ in G24-8b(fC,dC) are:

$$\begin{aligned} &p_i(t) = 1,\ q_1(t) = -(c_1 + r_a * \cos(k\pi t)),\ q_2(t) = -(c_2 + r_a * \sin(k\pi t)) \\ &r_i(t) = 1,\ s_i(t) = 0,\ i = 1, 2 \end{aligned} \tag{4.26}$$

where $c_1 = 1.470561702$, $c_2 = 3.442094786232$ and $r_a = 0.858958496$.

In the experiments, for all test functions, the number of changes is set to 12 (i.e., t changes from 0 to 11), the change severity is medium (i.e., $k = 0.5$, and $S = 20$), and the change frequency is set to 500, 1000, 2000 fitness evaluations (FEs), respectively.

To evaluate the efficacy of SELS, 6 state-of-the-art algorithms were used for comparisons. They are, dGArepairRIGA [92], dGArepairHyperM [92], GSA + Repair [96], DDECV + Repair [5], DDECV[4] and EBBPSO-T [16]. Note that the first 4 of the compared algorithms use a repair scheme, so they need a lot of feasibility checking but they ignore the cost. To make a fair comparison, we run SELS only evaluating the feasibility for an infeasible solution and do not count in the number of used fitness evaluations. The resulting algorithm is denoted as eSELS and compared to the 4 repair algorithms. When compared to DDECV and EBBPSO, SELS evaluates both the feasibility and objective function value for every individual, no matter whether it is feasible or infeasible, which is the same to what DDECV and EBBPSO do.

To evaluate the performance of each algorithm, the modified offline error [92] averaged at every function evaluation is used. The modified offline error averaged at every function evaluation is defined as follows:

$$E_{MO} = \frac{1}{num\_of\_eval} \sum_{j=1}^{num\_of\_eval} e_{MO}(j) \tag{4.27}$$

where num_of_eval denotes the maximum number of function evaluations, and $e_{MO}(j)$ denotes the error of the best feasible solution obtained at j-th evaluation. If there are no feasible solutions at the j-th evaluation, the worst possible value that a feasible solution can have will be taken. The error value of a feasible solution means the difference between

its function value and the best possible value that a feasible solution can have. The smaller the modified offline error is, the better the algorithm performs. Here, the best possible value and the worst possible value that a feasible solution can have were approximated by experiments for each test function.

In the experiments, for both SELS and eSELS, intermediate crossover with $p_c = 1.0$, and Gaussian mutation with $p_m = 1/D$ and $scale = 0.1$ are used, respectively. In the mutation, at least one variable is mutated every time. The number of LS objective function evaluations (ls_{num}) is set to 16, and 4 individuals are used for change detection every generation. Each algorithm is run 50 times on each test function.

4.3.2 Comparison Results with Existing Algorithms on 1000 FEs

Table 4.1 summarises the mean and standard deviation of the modified offline error values over 50 runs obtained by DDECV, EBBPSO-T and SELS as well as the performance rank of each algorithm on each test function (in case of ties, average ranks are assigned) based on the mean values. We applied the Friedman test and further a Holm's post-hoc procedure [32], which was used for multiple comparison of algorithms, to investigate whether SELS performed best on the set of test functions. The analysis shows that SELS has a significant improvement over DDECV and EBBPSO-T at the confidence level of 0.05. For test functions G24-3 and G24-7, SELS performed worse than EBBPSO-T. This is because that the global optima of them do not move to another feasible region but stay in the same feasible region as time goes.

Table 4.2 gives the performance rank of eSELS and the other 4 repair algorithms on each test function. We also applied the Friedman test and further a Holm's post-hoc procedure [32] to do a multiple-problem comparison among the 5 algorithms. The statistical test results show that eSELS performed significantly better than any other algorithm on the test function set at the confidence level of 0.05.

Table 4.1: Comparison results between DDECV, EBBPSO-T and SELS based on experimental results of DDECV and EBBPSO-T in their original papers [4] and [16], respectively. The best result obtained on each function is marked in **bold**.

Func	DDECV[rank]	EBBPSO-T[rank]	SELS[rank]
G24-l	0.109±0.033[3]	0.084±0.041[2]	**0.025±0.008**[1]
G24-2	0.126±0.030[2]	0.136±0.013[3]	**0.050±0.015**[1]
G24-3	0.057±0.018[3]	**0.032±0.005**[1]	0.044±0.022[2]
G24-3b	0.134±0.033[3]	0.104±0.015[2]	**0.052±0.018**[1]
G24-4	0.131±0.032[2]	0.138±0.022[3]	**0.082±0.021**[1]
G24-5	0.126±0.030[2.5]	0.126±0.019[2.5]	**0.054±0.014**[1]
G24-6a	0.215±0.067[3]	0.116±0.099[2]	**0.055±0.009**[1]
G24-6c	0.128±0.025[2]	0.251±0.061[3]	**0.052±0.008**[1]
G24-6d	0.288±0.055[2]	0.312±0.203[3]	**0.041±0.007**[1]
G24-7	0.106±0.022[3]	**0.045±0.009**[1]	0.087±0.016[2]
G24-8b	0.151±0.058[2]	0.312±0.086[3]	**0.055±0.022**[1]

Table 4.2: Comparison results among eSELS and the other 4 repair algorithms based on the experimental results in their original papers.

Func	dRepairRIGA[rank]	dRepairHyperM[rank]	GSA+Repair[rank]	DDECV+Repair[rank]	eSELS[rank]
G24-l	0.082±0.015[3]	0.093±0.023[4]	0.132±0.015[5]	0.061±0.010[2]	**0.013±0.007**[1]
G24-2	0.162±0.021[3]	0.171±0.026[4]	0.182±0.019[5]	0.062±0.006[2]	**0.030±0.008**[1]
G24-3	0.029±0.004[4]	0.027±0.005[2]	0.028±0.004[3]	0.046±0.006[5]	**0.018±0.004**[1]
G24-3b	0.058±0.007[2]	0.071±0.014[3]	0.076±0.009[4]	0.084±0.006[5]	**0.021±0.004**[1]
G24-4	0.140±0.028[5]	0.059±0.010[2]	0.073±0.012[3]	0.088±0.011[4]	**0.036±0.009**[1]
G24-5	0.152±0.017[4]	0.131±0.019[3]	0.153±0.013[5]	0.078±0.008[2]	**0.027±0.024**[1]
G24-6a	0.366±0.033[5]	0.358±0.049[4]	**0.033±0.003**[1]	0.036±0.005[2]	0.038±0.006[3]
G24-6c	0.323±0.037[4]	0.326±0.047[5]	0.045±0.004[3]	0.041±0.010[2]	**0.040±0.007**[1]
G24-6d	0.315±0.029[5]	0.286±0.035[4]	0.037±0.007[2]	0.079 ±0.006[3]	**0.029±0.004**[1]
G24-7	0.154±0.031[5]	0.067±0.014[3]	**0.018±0.002**[1]	0.107±0.011[4]	0.035±0.045[2]
G24-8b	0.341±0.053[5]	0.257±0.042[4]	0.192±0.034[3]	0.074±0.025[2]	**0.025±0.006**[1]

4.3.3 Comparison Results on 500 FEs and 2000 FEs

In the experiments, we also tested the performance of SELS and eSELS on the 11 test functions with the change frequency of 500 FEs and 2000 FEs. As only DDECV and DDECV + Repair have complete results on the change frequency of 500 FEs and 2000 FEs, the peformance of SELS and eSELS are compared to that of DDECV and DDECV + Repair, respectively.

Tables 4.3 and 4.4 show the mean and standard deviation of the modified offline error values over 50 runs obtained by SELS, eSELS, DDECV and DDECV + Repair on each test function under each change frequency of 500 FEs and 2000FEs. The better results obtained on each test function are marked in **bold**. It can be seen that SELS and eSELS performed better than DDECV and DDECV + Repair on more than 9 of 11 test functions, respectively. This demonstrates the robustness of SELS and eSELS to the change frequency of DCOPs.

Table 4.3: Comparison results between DDECV and SELS based on experimental results of DDECV in the original paper [4] under change frequency of 500 FEs and 2000 FEs. The better results obtained on each function are marked in **bold**.

F	Func	500 FEs		2000 FEs	
		DDECV	SELS	DDECV	SELS
G24-1	G24-1(dF,fC)	0.227±0.067	**0.068±0.028**	0.066±0.018	**0.011±0.003**
G24-2	G24-2(dF,fC)	0.162±0.032	**0.095±0.018**	0.071±0.016	**0.025±0.009**
G24-3	G24-3(dF,dC)	**0.087±0.024**	0.101±0.018	0.032±0.008	**0.024±0.009**
G24-3b	G24-3b(dF,dC)	0.225±0.070	**0.119±0.032**	0.078±0.015	**0.023±0.005**
G24-4	G24-4(dF,dC)	0.233±0.081	**0.143±0.031**	0.073±0.014	**0.053±0.013**
G24-5	G24-5(dF,dC)	0.195±0.033	**0.093±0.015**	0.081±0.011	**0.033±0.009**
G24-6a	G24-6a(2DR,hard)	0.267±0.114	**0.110±0.014**	0.103±0.032	**0.030±0.003**
G24-6c	G24-6c(2DR,easy)	0.173±0.048	**0.112±0.019**	0.063±0.013	**0.030±0.004**
G24-6d	G24-6d(2DR,hard)	0.414±0.083	**0.081±0.014**	0.139±0.027	**0.021±0.004**
G24-7	G24-7(fF,dC)	0.156±0.038	**0.132±0.020**	0.062±0.014	**0.047±0.009**
G24-8b	G24-8b(fC,OICB)	0.332±0.108	**0.113±0.030**	0.078±0.032	**0.025±0.016**

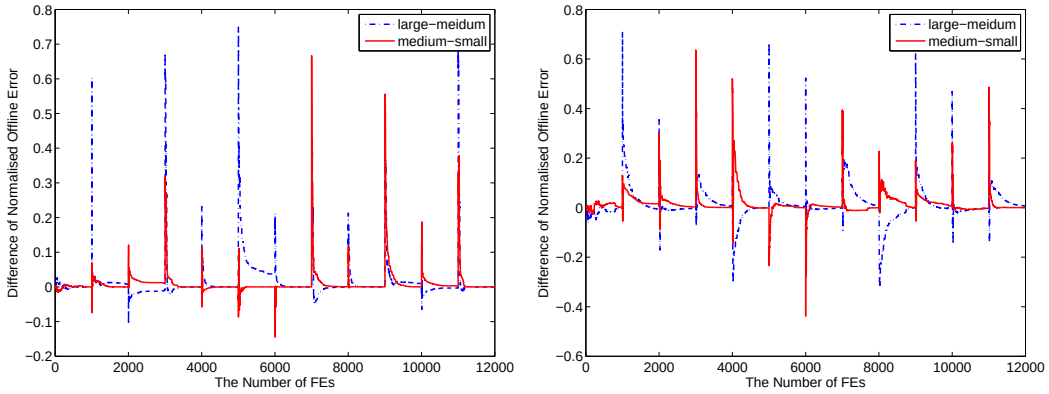
4.3.4 The Performance Effect of Change Severity and ls_{num}

To evaluate the performance of SELS on different changes, we also conducted experiments on small change severity (i.e., $k = 1.0$ and $S = 10$) and large severity (i.e., $k = 0.25$

Table 4.4: Comparison results between DDECV+Repair and eSELS based on experimental results of DDECV+repair in the original paper [5] under change frequency of 500 FEs and 2000 FEs. The better results obtained on each function are marked in **bold**.

F	Func	500 FEs		2000 FEs	
		DDECV+Repair	eSELS	DDECV+Repair	eSELS
G24-1	G24-1(dF,fC)	0.117±0.015	**0.032±0.015**	0.036±0.010	**0.005±0.002**
G24-2	G24-2(dF,fC)	0.137±0.008	**0.070±0.018**	0.035±0.007	**0.018±0.006**
G24-3	G24-3(dF,dC)	0.062±0.008	**0.041±0.015**	0.036±0.002	**0.009±0.003**
G24-3b	G24-3b(dF,dC)	0.147±0.012	**0.050±0.014**	0.063±0.009	**0.010±0.003**
G24-4	G24-4(dF,dC)	0.143±0.012	**0.100±0.090**	0.057±0.005	**0.021±0.004**
G24-5	G24-5(dF,dC)	0.140±0.014	**0.094±0.078**	0.041±0.004	**0.016±0.005**
G24-6a	G24-6a(2DR,hard)	0.083±0.016	**0.075±0.012**	0.020±0.004	0.020±0.003
G24-6c	G24-6c(2DR,easy)	0.090±0.011	**0.084±0.013**	0.022±0.004	0.021±0.004
G24-6d	G24-6d(2DR,hard)	0.196±0.015	**0.058±0.009**	0.044±0.003	**0.015±0.002**
G24-7	G24-7(fF,dC)	**0.134±0.023**	0.193±0.164	0.057±0.005	**0.020±0.004**
G24-8b	G24-8b(fC,OICB)	0.189±0.033	**0.052±0.014**	0.041±0.012	**0.018±0.006**

and $S = 50$). Figure 4.1 gives the evolutionary curves of the normalised offline error differences between medium and small severity, and between large and small severity on two representative functions, G24-2 and G24-8b. The X axis denotes the number of objective function evaluations, and the Y axis denotes the difference of the normalised offline error at each evaluation, which is normalised on each problem in one test function. In general, we found that SELS performed best on small severity, second best on medium, and worst on large severity.



(a) G24-2 (b) G24-8b

Figure 4.1: The evolutionary difference curves of SELS between different change severity

We also conducted experiments to study the sensitivity of SELS to the parameter ls_{num} (the maximum fitness evaluations permitted for each local search). Six settings of

ls_{num} (4, 8, 12, 16, 20, 24) were tested and compared on the 11 test functions. The mean and standard deviation of the modified offline error obtained with each setting on the 11 test functions are shown in Table 4.5. It can be seen from Table 4.5 that the performance of SELS depends on the value of ls_{num}. The first 9 test functions prefer $ls_{num} = 16$, while the last two test functions prefer $ls_{num} = 24$.

Table 4.5: Comparison results among different ls_{num}. The best results achieved on each test function are marked in **bold**.

ls_{num}	G24-1	G24-2	G24-3	G24-3b
4	3.68e−02±1.20e−02	7.36e−02±1.43e−02	7.67e−02±1.51e−02	9.02e−02±1.60e−02
8	2.92e−02±8.70e−03	5.68e−02±1.19e−02	5.52e−02±1.37e−02	6.11e−02±1.32e−02
12	**2.73e−02±8.63e−03**	5.16e−02±1.23e−02	4.90e−02±1.80e−02	5.49e−02±1.47e−02
16	**2.49e−02±8.16e−03**	**5.00e−02±1.50e−02**	**4.41e−02±2.17e−02**	**5.19e−02±1.79e−02**
20	2.97e−02±9.04e−03	**4.60e−02±1.04e−02**	**4.07e−02±1.34e−02**	**5.08e−02±1.47e−02**
24	**2.55e−02±9.59e−03**	**4.91e−02±1.47e−02**	**4.08e−02±1.45e−02**	**4.94e−02±1.61e−02**
ls_{num}	G24-4	G24-5	G24-6a	G24-6c
4	1.20e−01±2.19e−02	8.09e−02±1.40e−02	9.36e−02±1.11e−02	9.17e−02±1.19e−02
8	9.18e−02±1.69e−02	6.08e−02±1.15e−02	6.44e−02±8.29e−03	6.27e−02±9.16e−03
12	9.84e−02±1.82e−02	**5.61e−02±1.07e−02**	7.49e−02±9.73e−03	7.50e−02±1.15e−02
16	**8.17e−02±2.09e−02**	**5.41e−02±1.35e−02**	**5.46e−02±9.09e−03**	**5.22e−02±7.57e−03**
20	1.03e−01±1.93e−02	**5.51e−02±1.24e−02**	8.15e−02±1.23e−02	7.83e−02±1.22e−02
24	1.11e−01±2.26e−02	**5.68e−02±1.22e−02**	8.72e−02±8.05e−03	8.90e−02±9.31e−03
ls_{num}	G24-6d	G24-7	G24-8b	∼
4	7.15e−02±1.02e−02	1.08e−01±1.54e−02	1.40e−01±1.72e−02	∼
8	5.01e−02±9.34e−03	8.21e−02±1.13e−02	8.12e−02±1.55e−02	∼
12	5.50e−02±7.62e−03	8.90e−02±1.05e−02	6.13e−02±1.67e−02	∼
16	**4.14e−02±6.68e−03**	8.65e−02±1.61e−02	5.53e−02±2.18e−02	∼
20	5.85e−02±9.10e−03	9.06e−02±1.15e−02	**4.46e−02±1.30e−02**	∼
24	6.59e−02±1.09e−02	**7.03e−02±1.12e−02**	**4.32e−02±1.80e−02**	∼

4.3.5 The Performance Effect of AM and LS

We further conducted experiments to check whether the use of AM and LS really help in SELS, and comparisons were made among (1) SELS without AM or LS (SELS-am-ls for short), (2) SELS without LS (SELS-ls for short), and (3) SELS.

Table 4.6 summarises the mean and standard deviation of the modified offline error for each of them along with the comparison results. The Wilcoxon rank-sum test at a confidence level of 0.05 was applied to compare them on each test function. It can be seen

from Table 4.6 that SELS-ls overall outperformed SELS-am-ls, and the used LS further improves SELS-ls. This demonstrates the benefits of using AM and LS.

Table 4.6: Comparison results among SELS-am-ls, SELS-ls, and SELS based on experimental results implemented on DCOP test functions. Here, +, −, and ≈ denote whether one algorithm is better, worse or equal to another according to Wilcoxon ranksum test with a level of 0.05.

Func	SELS-am-ls	SELS-ls vs SELS-am-ls	SELS vs SELS-ls
G24-1	0.133±0.037	0.069±0.019 +	0.025±0.008 +
G24-2	0.186±0.017	0.121±0.021 +	0.050±0.015 +
G24-3	0.124±0.038	0.118±0.025 ≈	0.044±0.022 +
G24-3b	0.233±0.039	0.144±0.021 +	0.052±0.018 +
G24-4	0.199±0.027	0.171±0.030 +	0.082±0.021 +
G24-5	0.162±0.022	0.117±0.017 +	0.054±0.014 +
G24-6a	0.318±0.040	0.163±0.020 +	0.055±0.009 +
G24-6c	0.284±0.030	0.152±0.017 +	0.052±0.008 +
G24-6d	0.198±0.033	0.128±0.023 +	0.041±0.007 +
G24-7	0.121±0.017	0.138±0.018 −	0.087±0.016 +
G24-8b	0.387±0.044	0.242±0.031 +	0.055±0.022 +

4.4 Chapter Summary

This chapter applied niching methods to address DCOPs and study the question of *whether niching method can help better solve DCOPs*. To answer this question, this chapter introduced a novel speciation-based method, SELS, to deal with DCOPs. SELS employs deterministic crowding (DC) [80], assortative mating (AM) [28], diversity introducing and feasibility rules to locate and track multiple feasible regions. Additionally, SELS uses a local search strategy to promote exploitation of the promising regions to quickly find the new optimum. The experimental studies in Section 4.3 demonstrated that SELS generally outperformed the state-of-the-art algorithms on a DCOP benchmark. The performance effects of AM and LS on DCOPs were also studied by experiments and the experimental results validated the benefits of using AM and LS in SELS.

CHAPTER 5

COMPETITIVE CO-EVOLUTION FOR FAST-CHANGING DYNAMIC CONSTRAINED OPTIMISATION

The previous chapter aims to address DCOPs through the use of niching EAs which decompose the problem based on the search space. In this chapter, we focus on fast-changing DCOPs and propose to deal with them from a different consideration. The proposed method decomposes a DCOP into two antagonistic sub-components, environment and solution, and applies competitive co-evolution (ComC) to co-evolve the two sub-components. The goal is to find a set of solutions in an offline way that are good for sub-problems in a DCOP. After this, the solution set is used to optimise the corresponding DCOP online by using the solution set for initialisation each time a change is detected. During the online optimisation, the solution set is updated according to the encountered environments. In Section 5.1, the motivation for this work is given. Section 5.2 will detail the proposed method. In Section 5.3, the experiments on dynamic constrained optimisation benchmark problems will be presented. Finally, Section 5.4 will conclude this chapter.

5.1 Motivation

DCOPs is a sub-category of dynamic optimisation problems (DOPs), for which an optimiser is expected to efficiently identify the new optimum once the problem has changed. To address DCOPs, researchers have proposed some methods mainly by making modifications to existing dynamic optimisation methods. Concretely, most of the modifications are made on the diversity-driven approaches. The authors in [96] employed the repair method to accept diversified infeasible individuals. To achieve a good trade-off between global search and local search, a diversity-based balancing strategy was proposed in [16]. In the previous chapter, we applied a speciation method to maintain good solutions in different feasible regions during the optimisation process. The authors in [13] proposed to locate and track feasible regions by combining a multi-population approach and a gradient-based repair method.

Being able to efficiently react to the change is one of the most desirable features of an algorithm for DOPs. In the context of DCOPs, this might be even more important since in a real-world application, getting a feasible solution in a hard time budget might be more important than finding the global optimum. For example, a dynamic service composition process must satisfy users' requirements within very short time limits or it becomes impractical [122]. In such cases, people might even be willing to sacrifice the solution quality (in terms of the objective function value) for efficiency. However, as discussed in Section 1.3.3, none of the above-mentioned methods for DCOPs can well deal with these scenarios, i.e., DCOPs with very fast changes.

In previous studies on DOPs and DCOPs, it is usually implicitly assumed that the investigated algorithm is applied to DOPs or DCOPs from scratch, i.e., their initial solutions are randomly generated. In contrast, in most real-world scenario, one may have pretty long time to improve a system (and the algorithm behind it) before putting it into full use. For example, a company may spend months to polish its backend system before launching a new service composition system, and once the system is put online, it may need to react to changes on a daily basis.

Inspired from this, suppose we can maintain a set of solutions, finding them before going online and updating them online, and for any possible change induced by the environments, one of the solutions can be efficiently modified to get a good solution. Then, the DOP or DCOP in the online phase would not be as challenging as we expected. However, this issue was largely overlooked in the previous studies on DOPs and DCOPs, and thus motivated the research questions concerned in this chapter, namely, *whether identifying a set of promising solutions offline and updating them online would be beneficial to solve DCOPs online and how to achieve such solutions*, as posed in Section 1.3.3.

The first issue is non-trivial. Ideally, it would be very good if the optimal solution for each sub-problem of a DCOP can be found beforehand. A sub-problem for a DCOP denotes the optimisation under a fixed environment, which is a static problem. However, what is the environmental variable value for each sub-problem is hard to know in advance. Instead, one can search the set by finding the optimal solution with respect to every possible environmental parameter value. But, this process is time intractable. An alternative method is to sample the environments. To have a solution set of good coverage on all environments, the sampled environment each time should be the one that challenges the current solution set most. Moreover, the solution set needs to be updated to conquer the newly sampled environments. The relationship between the solution set and environments in this way is similar to the relationship between the host population and the parasite population in competitive co-evolution (ComC). Motivated from this, we propose in this paper to employ ComC to search a good solution set.

After finding a set of good solutions, the online optimisation will be started. Specifically, the found set will be used as initial solutions once a change is detected, local search strategy will be applied on the set to do further optimisation, and meanwhile the solution set is updated. The proposed method is called competitive co-evolution for dynamic constrained optimisation (CCDO). In the following sections, the CCDO method will be detailed first. Then, the experimental studies will be presented to answer the question of *whether identifying a set of promising solutions in advance would be beneficial to DCOPs*

and the question of *whether decomposing DCOPs into solution population and environment population and co-evolving them using ComC can find a better solution set* that are posed in Section 1.3.3.

5.2 The Proposed Approach

5.2.1 General Framework

Algorithm 10 gives the general framework for CCDO. ComC is applied to find a solution set for the sub-problems in a DCOP in an offline way. After finding a set of good solutions, the online optimisation will be started. Specifically, the found set will be used as initial solutions once a change is detected, local search strategy will be applied on the set to do further optimisation. In the following sub-sections, the two parts will be detailed.

Algorithm 10 General Framework for CCDO

Require:
The search space for the solution: $\boldsymbol{R}^{D_x}$
The range for the environmental parameters: $\boldsymbol{R}^{\boldsymbol{D_a}}$
The maximum number of generations: G_{max}
The number of individuals to detect changes: $detect_k$
The memory to store locally best soluiton: mem_{best}
1: Do competitive co-evolutionary search based on $\boldsymbol{R}^{D_x}$ and $\boldsymbol{R}^{\boldsymbol{D_a}}$ for G_{max} generations
2: Archive solution set $\boldsymbol{S}$ found by ComC according to Algorithm 13
3: Initialise a population *pop* based on $\boldsymbol{S}$
4: Evaluate *pop* with the current f
5: **while** stopping criteria is not satisfied **do**
6: Do local search on *pop* according to Algorithm 14
7: Re-evaluate $detect_k$ sentinel solutions to detect changes
8: **if** change is detected **then**
9: Update the solution set $\boldsymbol{S}$
10: Re-intialise the population according to Algorithm 15
11: **end if**
12: **end while**

5.2.2 Competitive Co-evolutionary Search for Solution Set

The framework of ComC can be found in Algorithm 4 in Chapter 2. When employing ComC to search the solution set, the host population is used to represent the solution set and the parasite population is used to represent the environmental parameters. The host population is denoted as SP (solution population) and the parasite population is denoted as EP (environment population). Figure 5.1 shows the representation for SP and EP.

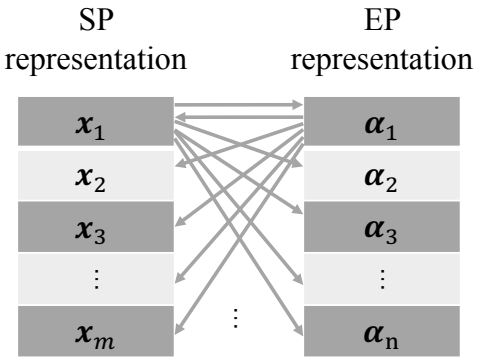


Figure 5.1: Representation for SP and EP

In SP, each individual is denoted as $\boldsymbol{x}_i (i = 1, 2, ..., m)$. In EP, each individual is denoted as $\boldsymbol{\alpha}_j (j = 1, 2, ..., n)$. Each $\boldsymbol{x}_i$ represents a candidate solution for the problem. Each $\boldsymbol{\alpha}_j$ represents a fixed environment and the values of $\boldsymbol{\alpha}_j$ denote the values of environmental parameters. The problem under the environment of $\boldsymbol{\alpha}_j$ is a static problem. The sizes of SP and EP are recorded as m and n, respectively. Each $\boldsymbol{x}_i$ is a candidate solution for the dynamic problem under the environment of $\boldsymbol{\alpha}_j$. The evolution of SP is to obtain better and better solutions for individuals in EP and the evolution of EP is to obtain more and more challenging environments for individuals in SP. The evaluation of each individual in SP (EP) depends on the individuals in EP (SP). The lines with an arrow are used to show such a testing and being tested relationship between SP and EP.

As discussed in Section 5.1, we would like to find a solution set that has a good overall coverage on all environments. Thus, solutions in SP should not only conquer more and more challenging environments in EP but also maintain good performance on previously conquered environments. To keep good performance of SP on previously well addressed

environments, we introduce an external archive (denoted as EAr) to store individuals in EP that SP peforms well on in previous generations. When evaluating SP, both individuals in EP and EAr are considered.

Evolution of SP

The evolution of SP is to obtain a good solution set for a dynamic problem under environments represented by individuals in EP and EAr. Thus, the fitness of an individual in SP depends on how much it can help SP in solving individuals in EP and EAr. In this work, the fitness of $\boldsymbol{x}_i$ in SP is defined as the drop of the performance of SP on EP and EAr after deleting $\boldsymbol{x}_i$ from SP.

Considering a dynamic constrained optimisation problem defined as in Eq.(2.3), $f(\boldsymbol{x}, \boldsymbol{\alpha})$ denotes the objective function under the environment of $\boldsymbol{\alpha}$. Here, the performance of SP on the dynamic problem under the environment $\boldsymbol{\alpha}$ is denoted as $F(SP, \boldsymbol{\alpha})$, which is defined as the fitness of $\boldsymbol{x}_{best}$ that performs best among all $\boldsymbol{x}_i$ in SP under the environment of $\boldsymbol{\alpha}$. If $\boldsymbol{x}_{best}$ is feasible, then $F(SP, \boldsymbol{\alpha}) = f(\boldsymbol{x}_{best}, \boldsymbol{\alpha})$; otherwise, $F(SP, \boldsymbol{\alpha})$ is set to the sum of constraint violation values of $\boldsymbol{x}_{best}$ on $\boldsymbol{\alpha}$.

Considering the problem scaling, we define the fitness of $\boldsymbol{x}_i$ in SP as the number of $\boldsymbol{\alpha}$ in EP and EAr on which the performance changes if deleting $\boldsymbol{x}_i$ from SP. That is:

$$\begin{aligned} & Fit(\boldsymbol{x}_i, SP, EP \cup EAr) \\ = & \sum_{\boldsymbol{\alpha} \in EP \cup EAr} [F(SP, \boldsymbol{\alpha}) \neq F(SP/\boldsymbol{x}_i, \boldsymbol{\alpha})] \end{aligned} \tag{5.1}$$

When comparing two individuals in SP that have the same $Fit(\boldsymbol{x}_i, SP, EP \cup EAr)$, one individual is randomly selected.

Algorithm 11 shows the evolutionary process of SP for one generation. At every generation g, two individuals $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are randomly selected from SP at the current generation (i.e. SP_g). Then, a new individual $\boldsymbol{x}_{new}$ is generated by doing evolutionary operation on $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ and added to SP_g to get SP'_g. After this, the fitness of all

individuals in SP'_g will be evaluated using the fitness function defined in Eq. (5.1). The worst individual $\boldsymbol{x}_{worst}$ that has the smallest fitness will be deleted from SP'_g and the remaining individuals in SP_g will go to next generation as SP_{g+1}. If more than one individual has the smallest fitness, $\boldsymbol{x}_{worst}$ is selected randomly from them.

Algorithm 11 SP Evolution for One Generation

1: Evaluate the fitness of each $\boldsymbol{x}$ in SP_g based on $EP_g \cup EAr$
2: Randomly select $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ from SP_g
3: Generate $\boldsymbol{x}_{new}$ based on $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ using mutation and crossover
4: Set $SP'_g = SP_g \cup \boldsymbol{x}_{new}$
5: Evaluate the fitness of each $\boldsymbol{x}$ in SP'_g
6: Set $SP_{g+1} = SP'_g / \boldsymbol{x}_{worst}$

Evolution of EP

The evolution of EP is to obtain more and more challenging environments for individuals in SP. Thus, if individuals in SP perform better on a dynamic problem under the environment $\boldsymbol{\alpha}$ in EP, then the environment $\boldsymbol{\alpha}$ is less challenging and the fitness of $\boldsymbol{\alpha}$ should be lower. To evaluate the fitness of an individual $\boldsymbol{\alpha}_j$ in EP, we randomly generate a set of solutions (denoted as I) and use the improvement obtained by SP over I on $\boldsymbol{\alpha}_j$ to evaluate it. According to whether the best solutions for $\boldsymbol{\alpha}_j$ in I and SP are feasible and whether an improvement is investigated, an individual $\boldsymbol{\alpha}_j$ can be categorised into the following four cases:

1. The best solution for $\boldsymbol{\alpha}_j$ in SP is infeasible. In this case, the farther the best solution is from the feasible boundary, the more challenging $\boldsymbol{\alpha}_j$ is. Thus, the fitness of $\boldsymbol{\alpha}_j$ is defined as the sum of constraint violation values of the best solution.

2. The best solution for $\boldsymbol{\alpha}_j$ in SP is feasible, but it is worse than the best solution for $\boldsymbol{\alpha}_j$ in I. In this case, the improvement obtained by I over SP on $\boldsymbol{\alpha}_j$ is used to evaluate it. That is,

$$Fit(\boldsymbol{\alpha}_j) = \frac{\min_{\boldsymbol{x}\in SP} F(x, \alpha_j) - \min_{\boldsymbol{x}\in I} F(x, \alpha_j)}{\max(|\min_{\boldsymbol{x}\in I} F(x, \alpha_j)|, |\min_{\boldsymbol{x}\in SP} F(x, \alpha_j)|)} \tag{5.2}$$

The larger the improvement is, the more challenging $\boldsymbol{\alpha}_j$ is.

3. The best solution for $\boldsymbol{\alpha}_j$ in SP is feasible and the best solution in I is infeasible. In this case, the improvement is hard to calculate. In this work, the fitness of $\boldsymbol{\alpha}_j$ is defined as the function value of the best solution in SP for $\boldsymbol{\alpha}_j$ (i.e. $\min_{\boldsymbol{x}\in SP} F(x, \alpha_j)$).

4. Both the best solutions for $\boldsymbol{\alpha}_j$ in SP and I are feasible. $\boldsymbol{\alpha}_j$ is considered less challenging if a larger improvement is obtained by SP over I. Thus, the fitness of $\boldsymbol{\alpha}_j$ is defined as the minus of the improvement obtained by SP over I on $\boldsymbol{\alpha}_j$. That is,

$$Fit(\boldsymbol{\alpha}_j) = \frac{\min_{\boldsymbol{x}\in SP} F(x, \alpha_j) - \min_{\boldsymbol{x}\in I} F(x, \alpha_j)}{\max(|\min_{\boldsymbol{x}\in I} F(x, \alpha_j)|, |\min_{\boldsymbol{x}\in SP} F(x, \alpha_j)|)} \tag{5.3}$$

For each case, the larger the fitness is, the more challenging $\boldsymbol{\alpha}_j$ is. When comparing individuals belonging to different cases, an individual that belongs to case 1 is considered more challenging than an individual that belongs to case 2, an individual belonging to case 2 is considered more challenging than an individual belonging to case 3 and an individual belonging to case 4. When comparing an individual belonging to case 3 and an individual belonging to case 4, a random one is selected.

Algorithm 12 gives the evolutionary process of EP for one generation. At every generation g, a new population EP_{new} is generated based on EP_g using evolutionary operation. Then, EP_g and EP_{new} are combined to obtain EP'_g and the fitness of individuals in EP'_g will be evaluated with the above-mentioned method. After this, EP_g and EP_{new} will be compared using a pair-wise comparison. All winners will enter next generation as EP_{g+1} and all losers will be used to update the external archive EAr. Here, losers mean that SP performs better on them. As SP is expected to perform well under all environments, we would like to maintain the performance of SP on these losers while improving SP to conquer the winners. Thus, all losers are used to update the archive which is used to evaluate SP together with the winners.

Algorithm 12 EP Evolution for One Generation

1: Generate a new population EP_{new} based on EP_g using crossover and mutation
2: Set $EP_g = EP_g \cup EP_{new}$
3: Evaluate each α in $EP_g^{'}$
4: **for** i=1,2,...,$size\_of\_EP$ **do**
5: **if** EP_g^i is more challenging than EP_{new}^i **then**
6: $EP_{g+1}^i = EP_g^i$
7: $EP_{worse}^i = EP_{new}^i$
8: **else**
9: $EP_{g+1}^i = EP_{new}^i$
10: $EP_{worse}^i = EP_g^i$
11: **end if**
12: **end for**
13: Using EP_{worse} to update the archive EAr

To Maintain and Update External Archive

The external archive EAr is to store previously good environments. In the evolution of EP, the individuals that do not enter next generation will be saved into EAr as they are less challenging. At the beginning, all $\boldsymbol{\alpha}$ individuals that do not enter next generation will be saved directly into EAr. When the number of added individuals achieves the size of EAr, each new individual $\boldsymbol{\alpha}$ will be added according to the following replacement rules:

1. Set $EAr' = EAr \cup \boldsymbol{\alpha}$,

2. Calculate the diversity contribution of each individual in EAr',

3. Delete the individual that has the least contribution.

The replacement rules delete the individuals that contribute least to the diversity of the archive. This is to maintain the diversity of the archive. In this work, the diversity contribution of each individual is calculated as the reduction of the diversity of the archive if deleting this individual. The diversity of the archive is calculated as the mean value of the difference between each pair of individuals in the archive. The difference of two individuals $\boldsymbol{\alpha}_i$ and $\boldsymbol{\alpha}_j$ is calculated as follows:

$$Diff(\boldsymbol{\alpha}_i, \boldsymbol{\alpha}_j) = -\text{Spearman}(Ranks(SP, \boldsymbol{\alpha}_i), Ranks(SP, \boldsymbol{\alpha}_j)) \tag{5.4}$$

where Spearman denotes the Spearman's rank correlation coefficient or Spearman's rho [114] between $Ranks(SP, \boldsymbol{\alpha}_i)$ and $Ranks(SP, \boldsymbol{\alpha}_j)$. $Ranks(SP, \boldsymbol{\alpha}_i)/Ranks(SP, \boldsymbol{\alpha}_j)$ denotes the performance rank vector of all individuals in SP on $\boldsymbol{\alpha}_i/\boldsymbol{\alpha}_j$. If $\boldsymbol{x}_i$ is the individual in SP that peforms best on $\boldsymbol{\alpha}$, then its rank is 1. If it performs second best, its rank is 2, and so on.

Competitive Co-evolutionary Search Algorithm

Algorithm 13 details the evolutionary process to search the solution set by ComC. In this algorithm, D_x and D_a are the dimension of the solution space and environment space for a DCOP, respectively. The solution population obtained in the final generation is used as the solution set on which to do online optimisation. The found solution set for a dynamic problem will be later used for the online optimisation to address the dynamic problem.

Algorithm 13 Competitive Co-evolutionary Search Algorithm

Require:
The search space for the solution: $\boldsymbol{R}^{D_x}$
The range for the enviromental parameters: $\boldsymbol{R}^{D_a}$
The maximum number of generations: G_{max}
The size for the initial solution set I: i_{size}
The maximum generation number to evolve SP_g: esp
Ensure:
The final solution population, $SP_{G_{max}}$
1: Set $g = 0$
2: Initialise solution population SP_0 based on $\boldsymbol{R}^{D_x}$
3: Initialise environment population EP_0 based on $\boldsymbol{R}^{D_a}$
4: Set $EAr = \varnothing$
5: **while** $g \leq G_{max}$ **do**
6: Evolve SP_g for esp generations to get SP_{g+1} based on $EP_g \cup EAr$ according to Algorithm 11
7: Randomly sample a solution set I with i_{size}
8: Evolve EP_g for one generation based on SP_{g+1} and I according to Algorithm 12
9: Update the archive EAr
10: Set $g = g + 1$
11: **end while**
12: Output $SP_{G_{max}}$

5.2.3 Online Optimisation

When the solution set is found by ComC, it will be used to optimise the corresponding DCOP online. Concretely, the found set will be used as the initial individuals at the beginning and each time the change is detected. During the interval between two consecutive changes, local search operations will be conducted on the individuals in the population to address the current sub-problem.

Local Search Strategy

In this work, we use sequential quadratic programming (SQP) [11] to do local search in the online optimisation. SQP is an iterative method which deals with a constrained nonlinear optimisation problem by solving a sequence of relatively simple optimisation sub-problems. Suppose a constrained nonlinear optimisation problem has the following formulation:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimise}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & g_i(\boldsymbol{x}) \leq 0, i = 1, 2, ..., k \\ & h_i(\boldsymbol{x}) = 0, j = 1, 2, ..., l \end{aligned} \tag{5.5}$$

where $\boldsymbol{x} = [x_1, x_2, ..., x_{D_x}]^T$ and D_x is the dimension of the decision space. Then, we can get the Lagrangian function for this problem as:

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T * G(\boldsymbol{x}) - \boldsymbol{\mu}^T * H(\boldsymbol{x}) \tag{5.6}$$

where $G(\boldsymbol{x}) = \{g_i(\boldsymbol{x}) | i = 1, 2, ..., k\}$, $H(\boldsymbol{x}) = \{h_i(\boldsymbol{x}) | i = 1, 2, ..., l\}$, $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, ..., \lambda_k]^T$ and $\boldsymbol{\mu} = [\mu_1, \mu_2, ..., \mu_l]^T$ are vectors of Lagrangian multipliers.

Suppose the solution point at the k-th iteration is $\boldsymbol{x}_k$, then the search direction $\boldsymbol{d}_k =$

$[d_1, d_2, ..., d_{D_x}]^T$ is a solution to the quadratic programming sub-problem as follows:

$$\begin{aligned} \underset{\boldsymbol{d}_k}{\text{minimise}} \quad & f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}_k + \frac{1}{2} \boldsymbol{d}_k^T \nabla_{xx}^2 L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \boldsymbol{d}_k \\ \text{s.t.} \quad & g_i(\boldsymbol{x}_k) + \nabla g_i(\boldsymbol{x}_k)^T \boldsymbol{d}_k \leq 0, i = 1, 2, ..., k \\ & h_i(\boldsymbol{x}_k) + \nabla h_i(\boldsymbol{x}_k)^T \boldsymbol{d}_k = 0, j = 1, 2, ..., l \end{aligned} \tag{5.7}$$

Here, each of $\nabla f(\boldsymbol{x}_k)$, $\nabla g_i(\boldsymbol{x}_k)$, and $\nabla h_j(\boldsymbol{x}_k)$ is estimated by the forward difference formula as follows:

$$\begin{aligned} \nabla f(\boldsymbol{x}_k) &= \frac{1}{\eta}[f(\boldsymbol{x}_k + \boldsymbol{e}_1) - f(\boldsymbol{x}_k), f(\boldsymbol{x}_k + \boldsymbol{e}_2) - f(\boldsymbol{x}_k), ..., f(\boldsymbol{x}_k + \boldsymbol{e}_{D_x}) - f(\boldsymbol{x}_k)]^T \\ \nabla g_i(\boldsymbol{x}_k) &= \frac{1}{\eta}[g_i(\boldsymbol{x} + \boldsymbol{e}_1) - g_i(\boldsymbol{x}), g_i(\boldsymbol{x} + \boldsymbol{e}_2) - g_i(\boldsymbol{x}), ..., g_i(\boldsymbol{x} + \boldsymbol{e}_{D_x}) - g_i(\boldsymbol{x})]^T \\ \nabla h_j(\boldsymbol{x}_k) &= \frac{1}{\eta}[h_i(\boldsymbol{x} + \boldsymbol{e}_1) - h_i(\boldsymbol{x}), h_i(\boldsymbol{x} + \boldsymbol{e}_2) - h_i(\boldsymbol{x}), ..., h_i(\boldsymbol{x} + \boldsymbol{e}_{D_x}) - h_i(\boldsymbol{x})]^T \end{aligned} \tag{5.8}$$

where η is a very small positive value (set as 1.49e-8 in this work) and $\boldsymbol{e}_i$ is a vector in which the i-th element is η and the other elements are all 0. The $\nabla_{xx}^2 L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ is a positive definite approximation of the Hessian matrix of $L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ on $\boldsymbol{x}$. It is initialised as a positive definite matrix, and updated based on $\nabla f(\boldsymbol{x})$, $\nabla g_i(\boldsymbol{x})$, and $\nabla h_j(\boldsymbol{x})$ using Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [1].

After $\boldsymbol{d}_k$ is found, $\boldsymbol{x}_{k+1}$ is set to $(\boldsymbol{x}_k + \boldsymbol{d}_k)$ and the $(k+1)$-th iteration starts. The starting point for the SQP iteration at the beginning is the solution on which local search is conducted. To implement SQP, the Matlab optimisation toolbox and *fmincon* function was used in this chapter. More implementation details can be found in [1].

At every generation, each individual in the population can undergo local search. However, doing local search on similar individuals can not bring improvements but cost more computational resources. To avoid conducting local search on similar individuals, individuals that have undergone local search are recorded in the memory, mem_{ls}. At every generation, if the distance between one individual and the nearest individual in the mem_{ls} is less than 0.01, local search will not be conducted on this individual. This strategy is similar to the adaptive local search strategy used in [13]. At the end of every generation,

individuals whose distance to the nearest individual in the mem_{ls} is less than 0.01 will be replaced by random individuals.

In addition to the SQP operation, the Gaussian mutation operator is also used to mutate every individual. The mutant individual will replace the current individual if it is better. Algorithm 14 shows the details of the local search process.

Algorithm 14 Local Search Process

for each solution $\boldsymbol{x}_i$ in the population pop_g **do**
  Re-evaluate $\boldsymbol{x}_i$ to detect whether a change happens
  if a change is detected **then**
    Re-intialise the population according to Algorithm 15
  else
    if the distance between $\boldsymbol{x}_i$ and mem_{ls} is larger than 1e-2 **then**
      Do SQP local search on $\boldsymbol{x}_i$ to get $\boldsymbol{x}_i^{best}$
      Archive $\boldsymbol{x}_i$ into mem_{ls}
      Archive $\boldsymbol{x}_i^{best}$ into mem_{best}
      Set $\boldsymbol{x}_i = \boldsymbol{x}_i^{best}$
    end if
    Generate $\boldsymbol{x}_i^{'} = \boldsymbol{x}_i + \delta * randn(1, D_x)$
    if $f(\boldsymbol{x}_i^{'})$ is better than $f(\boldsymbol{x}_i)$ **then**
      Set $\boldsymbol{x}_i = \boldsymbol{x}_i^{'}$
    end if
  end if
end for

Change Detection

At each generation, individuals in the current population will be re-evaluated to detect whether a change happens before local search operation. In some situations, the population might converge and this detection strategy can not detect changes that happen in other regions. Considering this, we generate several random individuals and use them as sentinels. After each local search operation, the randomly generated individuals will also be re-evaluated to check whether there is a change.

Update Mechanism

As the environments of a DCOP that do appear in the online optimisation are of top concern, the solution set obtained by ComC should be updated according to the encountered environments before being used as initial solutions. One direct way to implement this to include every newly encountered environment into the environment population EP or the archive EAr and run ComC online to update the solution set $\boldsymbol{S}$. Each time a change is detected, the current solutions in the solution population SP are picked out for intialisation. However, running ComC online will cost extra computing resources. Considering this, we use a very simple mechanism to update the solutions set $\boldsymbol{S}$. That is, the best solution found in last environment will be added to $\boldsymbol{S}$ when a change is detected.

Re-initialisation Process

To deal with the DCOPs in which the optimum switches between feasible regions, the optimal solutions obtained by last local search operations are used as the initial individuals together with the solution set $\boldsymbol{S}$ when a change is detected. Algorithm 15 shows the details of the re-initialisation process.

Algorithm 15 Population Re-initialisation Process

Require:
The memory to store locally best soluiton: mem_{best}
The solution set $\boldsymbol{S}$ found by ComC
Set $pop_{initial} = \boldsymbol{S} \cup mem_{best}$
Set $pop = \emptyset$
for each $\boldsymbol{x}_i$ in $pop_{initial}$ **do**
  if the distance between $\boldsymbol{x}_i$ and pop is larger than 1e-2 **then**
    $pop = pop \cup \boldsymbol{x}_i$
  end if
end for
Do local search on pop according to Algorithm 14

5.3 Experimental Studies

To evaluate whether identifying a set of promising solutions offline would be beneficial to DCOPs, a DCOP benchmark was used to test the performance of CCDO and comparisons are made between it and two state-of-the-art dynamic constrained optimisation methods. At the cost of efforts spent in the competitive co-evolutionary search process, we expect CCDO to show a fast adaptation to DCOPs with fast changes. Therefore, different change frequencies of DCOPs were considered to test the performance of CCDO in the experimental studies.

5.3.1 Bechmark Problems

In the literature, there exist two suites of DCOP benchmark test functions, one was proposed in [94] and the other was proposed in [13]. The latter one contains test functions that have smaller feasible regions. However, DCOPs with small feasible regions will prefer methods that deal with constraints using a repair mechanism.

Considering this, only the DCOP test problems in the former suite were used in this study. In the experiments, 9 DCOP benchmark test functions proposed in [94] were used. They are G24-l (dF,fC), G24-2 (dF,fC), G24-3 (dF,dC), G24-3b (dF,dC), G24-4 (dF,dC), G24-5 (dF,dC), G24-6a (dF,fC), G24-6c (dF,fC), G24-7 (fF,dC). Here, 'dF', 'fF', 'dC', and 'fC' mean dynamic objective function, fixed objective function, dynamic constraint functions, and fixed constraint functions, respectively. All of them have the following form:

$$\begin{aligned} &\min_{\boldsymbol{x}} \ f(\boldsymbol{x}, \boldsymbol{\alpha}(t)) \\ &\text{subject to} : g_i(\boldsymbol{x}, \boldsymbol{\alpha}(t)) \leq 0, i = 1, 2, ..., k \end{aligned} \tag{5.9}$$

where $\boldsymbol{x}$ denotes the decision variable vector, k denotes the number of constraints, $\boldsymbol{\alpha}(t)$ is the vector of environmental parameters. They vary at a certain frequency as time goes by. Note that we assume in this chapter that the variation range of $\boldsymbol{\alpha}(t)$ is known beforehand.

Further details about them can be found in Section 4.3.1 of Chapter 4.

In the experiments, the change frequency for each test function is set to 100, 500 and 1000 objective function evaluations, respectively. The change severity is set to medium (i.e., $k = 0.5$, and $S = 20$). The environmental parameters for the 9 DCOP test functions are the variables in them that changes as time goes by. The details of the environmental parameters and their ranges in each test function are given in Table 5.1. The range for each environmental parameter is set as the interval between the minimum value and the maximum value that the parameter can have when the number of problem changes is set to 12.

Table 5.1: Environmental parameters for each test function

Function	Parameters	Ranges	Solution Size
G24-l (dF,fC)	p_1	[-1,1]	2
G24-2 (dF,fC)	p_1, p_2	[-1,1],[-1,1]	5
G24-6a (dF,fC)	p_1	[-1,1]	2
G24-6c (dF,fC)	p_1	[-1,1]	2
G24-3 (dF,dC)	s_2	[-0.2,2]	many
G24-3b (dF,dC)	p_1, s_2	[-1,1], [-0.2,2]	many
G24-4 (dF,dC)	p_1, s_2	[-1,1], [0,2.2]	many
G24-5 (dF,dC)	p_1, p_2, s_2	[-1,1],[-1,1],[0,2.2]	many
G24-7 (fF,dC)	s_2	[0,2.2]	many

When the value of environmental parameter changes, the optimal solution might change or not. When the environmental parameter changes in the range, an ideal solution set should be the one that owns an optimal solution for the sub-problem in any environment. Table 5.1 also gives the smallest size that an ideal solution set can have for each test function in the column of 'Solution size'. According to this, the 9 test functions can be classified into two groups. The first group includes G24-l (dF,fC), G24-2 (dF,fC), G24-6a (dF,fC), G24-6c (dF,fC), for which the smallest ideal solution set has a limited size. The second group includes G24-3 (dF,dC), G24-3b (dF,dC), G24-4 (dF,dC), G24-5 (dF,dC), G24-6d (dF,fC), and G24-7 (fF,dC), for which the smallest ideal solution set has an unlimited size.

5.3.2 Overview on Compared Algorithms

The proposed CCDO method was compared with two state-of-the-art algorithms, SELS [78] and LTFR-DSPSO [13]. Different from SELS, LTFR-DSPSO applied a gradient-based repair method [21] to handle the constraints. Moreover, all constraint calculation costs of LTFR-DSPSO were ignored in [13]. To make a fair comparison, the CCDO method was modified by using the same gradient-based repair method when compared to LTFR-DSPSO. The gradient-based repair process is as follows:

Suppose a constrained nonlinear optimisation problem has the same formulation as in Eq. (5.5). According to first-order Taylor expansion, we can have at the point $\boldsymbol{x}_0$:

$$\begin{aligned} g_i(\boldsymbol{x}) &\approx g_i(\boldsymbol{x}_0) + \nabla^T g_i(\boldsymbol{x}_0)(\boldsymbol{x} - \boldsymbol{x}_0), i = 1, 2..., k \\ h_j(\boldsymbol{x}) &\approx h_j(\boldsymbol{x}_0) + \nabla^T h_j(\boldsymbol{x}_0)(\boldsymbol{x} - \boldsymbol{x}_0), j = 1, 2..., l \end{aligned} \tag{5.10}$$

where ∇g_i and ∇h_j are estimated as in Eq. (5.8).

Let $GH(\boldsymbol{x}) = [g_1(\boldsymbol{x}), g_2(\boldsymbol{x}), ..., g_k(\boldsymbol{x}), h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), ..., h_l(\boldsymbol{x})]^T$, and $J_{GH} = [\nabla^T g_1(\boldsymbol{x}_0); \nabla^T g_2(\boldsymbol{x}_0); ...; \nabla^T g_k(\boldsymbol{x}_0); \nabla^T h_1(\boldsymbol{x}_0); \nabla^T h_2(\boldsymbol{x}_0); ...; \nabla^T h_l(\boldsymbol{x}_0)]$. According to Eq. (5.10), we can have:

$$GH(\boldsymbol{x}) \approx GH(\boldsymbol{x}_0) + J_{GH} * (\boldsymbol{x} - \boldsymbol{x}_0) \tag{5.11}$$

Let $\Delta g_i(\boldsymbol{x}) = \max(0, g_i(\boldsymbol{x}))$ and $\Delta h_i(\boldsymbol{x}) = \max(0, g_i(\boldsymbol{x}))$. Then, $\Delta GH(\boldsymbol{x}) = [\Delta g_1(\boldsymbol{x}), ..., \Delta g_k(\boldsymbol{x}), \Delta h_1(\boldsymbol{x}), ..., \Delta h_l(\boldsymbol{x})]$ is the constraint violation vector of $\boldsymbol{x}$.

Suppose $\boldsymbol{x}_0$ is infeasible, to repair $\boldsymbol{x}_0$ to a feasible solution $\boldsymbol{x}_1$ (i.e. $\Delta GH(\boldsymbol{x}_1) = \boldsymbol{0}$), we can have according to Eq. (5.11):

$$-\Delta GH(\boldsymbol{x}_0) = J_{GH} * (\boldsymbol{x}_1 - \boldsymbol{x}_0) \tag{5.12}$$

Then, we can get:

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - J_{GH}^{-1} * \Delta GH(\boldsymbol{x}_0) \tag{5.13}$$

When the matrix J_{GH} is not invertible, the inverse matrix J_{GH}^{-1} is approximated by the Moore-Penrose inverse in the experiments.

In this CCDO+Repair method, only the feasible solution and the sentinel solutions will be evaluated with the objective function. For each infeasible solution, a repair operation will be conducted. The repair-based local search process in CCDO+Repair is presented in Algorithm 16. If the solution obtained from local search is still infeasible, the repair operation is also conducted on this solution.

Algorithm 16 Repair-based Local Search Process

for each soluiton $\boldsymbol{x}_i$ in the population pop_g **do**
  Re-evaluate the feasibility of $\boldsymbol{x}_i$ to detect whether a change happens
  if a change is detected **then**
    Re-intialise the population according to Algorithm 15
  else
    if not isfeasible($\boldsymbol{x}_i$) **then**
      Repair $\boldsymbol{x}_i$ to $\boldsymbol{x}_i^{repair}$
      if $\boldsymbol{x}_i^{repair}$ is better than $\boldsymbol{x}_i$ **then**
        Set $\boldsymbol{x}_i = \boldsymbol{x}_i^{repair}$
      end if
    end if
    if the distance between $\boldsymbol{x}_i$ and mem_{ls} is larger than 1e-2 **then**
      Do SQP local search on $\boldsymbol{x}_i$ to get $\boldsymbol{x}_i^{best}$
      Archive $\boldsymbol{x}_i$ into mem_{ls}
      Archive $\boldsymbol{x}_i^{best}$ into mem_{best}
      if not isfeasible($\boldsymbol{x}_i^{best}$) **then**
        Repair $\boldsymbol{x}_i^{best}$ to $\boldsymbol{x}_i^{repair}$
        if $\boldsymbol{x}_i^{repair}$ is better than $\boldsymbol{x}_{best}$ **then**
          Set $\boldsymbol{x}_i^{best} = \boldsymbol{x}_i^{repair}$
        end if
      end if
      Set $\boldsymbol{x}_i = \boldsymbol{x}_i^{best}$
    end if
    Generate $\boldsymbol{x}_i^{'} = \boldsymbol{x}_i + \delta * randn(1, D_x)$
    if $f(\boldsymbol{x}_i^{'})$ is better than $f(\boldsymbol{x}_i)$ **then**
      Set $\boldsymbol{x}_i = \boldsymbol{x}_i^{'}$
    end if
  end if
end for

5.3.3 Performance Measures

For the SELS method and the LTFR-DSPSO method, different performance measures were considered to assess the efficacy of these two algorithms in their original papers [78, 13]. Considering this, different performance measures were also applied in this chapter when making different comparisons.

When comparing the CCDO with SELS, the modified offline error averaged at every function evaluation is used to evaluate the performance of each algorithm. The modified offline error averaged at every function evaluation is defined as follows:

$$E_{MO} = \frac{1}{num\_of\_eval} \sum_{j=1}^{num\_of\_eval} e_{MO}(j) \tag{5.14}$$

where num_of_eval denotes the maximum number of the function evaluations, and $e_{MO}(j)$ denotes the error of the best feasible solution obtained at j-th evaluation. If there are no feasible solutions at the j-th evaluation, the worst possible value that a feasible solution can have will be taken. The error value of a feasible solution means the difference between its function value and the best possible value that a feasible solution can have. The smaller the modified offline error is, then the better the algorithm performs.

When comparing CCDO+Repair with LTFR-DSPSO, the modified offline error averaged at every generation is used to evaluate the performance of each algorithm. The modified offline error averaged at every generation is defined as follows:

$$E_{MO} = \frac{1}{num\_of\_gen} \sum_{j=1}^{num\_of\_gen} e_{MO}(j) \tag{5.15}$$

where num_of_gen denotes the maximum number of the function evaluations, and $e_{MO}(j)$ denotes the error of the best feasible solution obtained at the j-th generation. If there are no feasible solutions at the j-th generation, the worst possible value that a feasible solution can have is taken.

5.3.4 Parameter Settings

Comparison with SELS

When compared to SELS, the number of changes is set to 12, which is the same as in [78]. The parameter setting for the competitive co-evolutionary search process and online optimisation is given in Table 5.2 and Table 5.3, respectively. The parameter values for SELS were set to the same as in its original paper. For the parameter setting of SQP, the other parameters were set as default except for the parameters mentioned in Table 5.3. The parameter setting for SELS is the same as in its original paper [78].

Table 5.2: Parameter settings for co-evolutionary search process when comparing CCDO with SELS

Parameter	Value
G_{max}	50
SP_{size}	10
EAr_{size}	10
EP_{size}	10
i_{size}	5
esp	50
SP evolution	Gaussian muation with scale = 0.1, rate = 0.5 Intermediate Crossover with rate = 0.5
EP evolution	Gaussian muation with scale = 0.05, rate = 0.5

Table 5.3: Parameter settings for online optimisation when comparing CCDO with SELS

Parameter	Value
$detect_k$	4
SQP	ConstraintTolerance = 0 HonorBounds = true MaxFunctionEvaluations = 20
Mutation	Gaussian muation with scale = 0.1

Comparison with LTFR-DSPSO

When compared to LTFR-DSPSO, the number of changes is set to 10, which is the same as in [13]. The parameter setting for the competitive co-evolutionary search process and online optimisation is given in Table 5.4 and Table 5.5, respectively. For the parameter setting of SQP, the other parameters are set as default except the parameters mentioned in Table 5.5. The experimental results of LTFR-DSPSO given in [13] were used for comparison.

Table 5.4: Parameter settings for co-evolutionary search process when comparing the proposed CCDO method with LTFR-DSPSO

Parameter	Value
G_{max}	50
SP_{size}	20
EAr_{size}	20
EP_{size}	20
i_{size}	5
esp	50
SP evolution	Gaussian muation with scale = 0.1, rate = 0.5 Intermediate Crossover with rate = 0.5
EP evolution	Gaussian muation with scale = 0.05, rate = 0.5

Table 5.5: Parameter settings for online optimisation when comparing the proposed CCDO method with LTFR-DSPSO

Parameter	Value
$detect_k$	4
SQP	ConstraintTolerance = 0 HonorBounds = true MaxFunctionEvaluations = 50
Mutation	Gaussian muation with scale = 0.1

5.3.5 Experimental Results

Before comparing the CCDO method and each of SELS and LTFR-DSPSO on the testing DCOPs, we first conducted experiments to check whether evolution of environment

population in the co-evolutionary search process can help find a set of solutions with a better coverage.

The Effectiveness of Environmental Evolution

In this experiment, we randomly generated the same number of environments at the beginning but fixed them during the co-evolutionary search process in Algorithm 13. The solution set found by this fixed process is compared to the solution set found by the co-evolutionary search process. The parameter setting for this experiments is set the same as in Table 5.2.

To evaluate the performance of the solution set, 50 sub-problems with randomly generated environmental parameter values were used as test sub-problems for each test function. The performance of the solution set on each sub-problem equals to the error value of the best solution in the solution set for this sub-problem. The error value of a feasible solution means the difference between its function value and the best possible value that a feasible solution can have. When there is no feasible solution in the solution set for a sub-problem, the worst possible objective function value that a feasible solution can have on this sub-problem will be taken. The average performance of the solution set on these 50 sub-problems over 50 runs was used for comparison. The Wilcoxon rank-sum test with a confidence level at 0.05 was used to make a comparison between the random method with fixed environments and the co-evolutionary method on each test function.

Table 5.6 summarises the mean and standard deviation of the average best error obtained on 50 sub-problems over 50 runs for each test function. The better results are marked in bold. It can be seen from Table 5.6 that the evolution of environments can help obtain a competitive or better solution set on the second group of test functions in which the smallest ideal solution set has an unlimited size. But, on the first group of test functions in which the smallest ideal solution set has a limited size, the random method with fixed environments performed better. The analysis about the reason is as follows.

For the first group of test problems, the coverage on all possible sub-problems can

be easily achieved by random environment sampling as the smallest ideal solution set has a limited size. As a comparison, the evolution of environments can not bring too much improvement on the coverage but needs more function evaluations. Consequently, the co-evolutionary method can not perform as well as the one without the evolution of environments.

For the second group of test problems, although the evolution of environments also costed more function evaluations, it is hard for several randomly generated environments to have a good coverage on this kind of problems. In this case, the evolution of environment population takes effect and thus the co-evolutionary method can show some advantages on this group of test problems.

Table 5.6: Comparison results between the co-evolutionary method and the random method with fixed environments. The better result obtained on each test function is marked in **bold**.

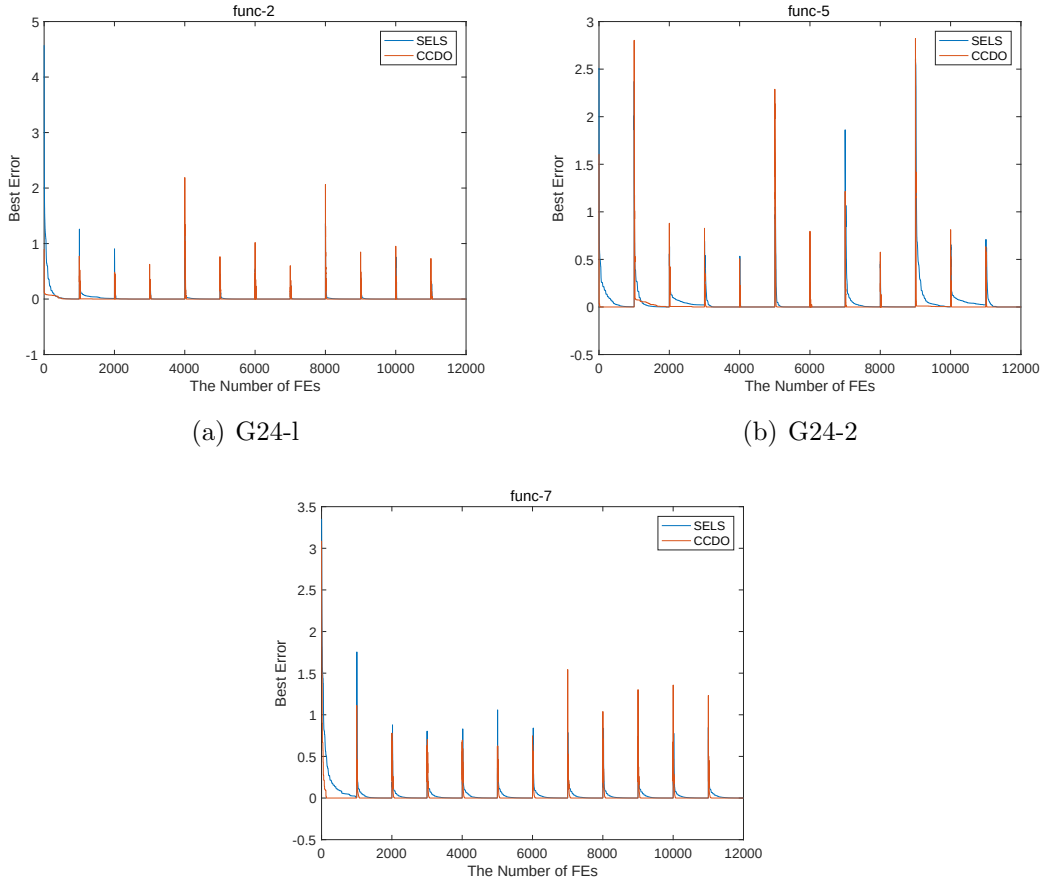
Func	Solution Size	Fixed Environments	Co-evolutionary
G24-1	2	**4.06e−02±6.25e−02**	7.98e−02±1.15e−01
G24-2	5	**5.19e−02±8.57e−02**	6.50e−02 ±8.64e−02
G24-6a	2	**4.52e−02±1.90e−01**	7.46e−02±2.63e−01
G24-6c	2	4.45e−02±4.51e−02	3.38e−02±3.21e−02
G24-3	many	6.46e−01±5.47e−01	**2.79e−01±3.14e−01**
G24-3b	many	6.03e−01±2.04e−01	5.85e−01±1.58e−01
G24-4	many	6.40e−01±2.14e−01	6.04e−01±1.75e−01
G24-5	many	2.86e−01±1.25e−01	2.81e−01±9.12e−02
G24-7	many	4.11e−01±1.97e−01	**2.50e−01±6.42e−02**

Comparison with SELS

Table 5.7 summarises the mean and standard deviation of the modified offline error over 50 runs obtained by the proposed CCDO method and SELS under the change frequency of 1000 FEs. We applied the Wilcoxon rank-sum test at a confidence level of 0.05 to test whether the CCDO method performed significantly better than SELS on each test function. In Table 5.7, the better result obtained on each test function is marked in bold. Figs. 5.2 and 5.3 plot the evolutionary curves for both CCDO and SELS.

Table 5.7: Comparison results between SELS and the CCDO method under the change frequency of 1000 FEs. The better result on each test function is marked in **bold**.

Func	SELS	CCDO
G24-1 (dF,fC)	2.49e−02±8.16e−03	**9.38e−03±7.65e−03**
G24-2 (dF,fC)	5.00e−02±1.50e−02	**1.53e−02±6.13e−03**
G24-3 (fF,dC)	4.41e−02±2.17e−02	**1.51e−02±7.36e−03**
G24-3b (dF,dC)	5.19e−02±1.79e−02	**3.86e−02±2.76e−02**
G24-4 (dF,dC)	**8.17e−02±2.09e−02**	1.21e−01±2.97e−02
G24-5 (dF,dC)	**5.41e−02±1.35e−02**	8.25e−02±2.08e−02
G24-6a (dF,fC)	5.46e−02±9.09e−03	**3.33e−02±7.69e−03**
G24-6c (dF,fC)	5.22e−02±7.57e−03	**3.07e−02±7.23e−03**
G24-7 (fF,dC)	8.65e−02±1.61e−02	8.26e−02±1.34e−02

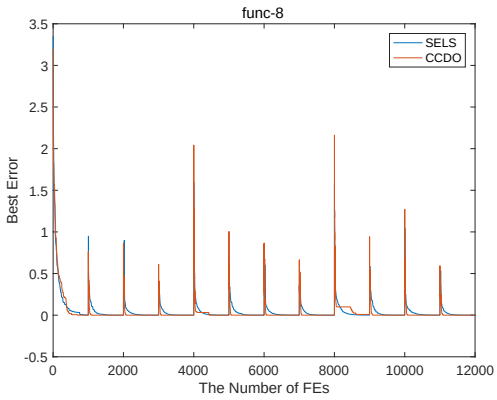


(a) G24-1

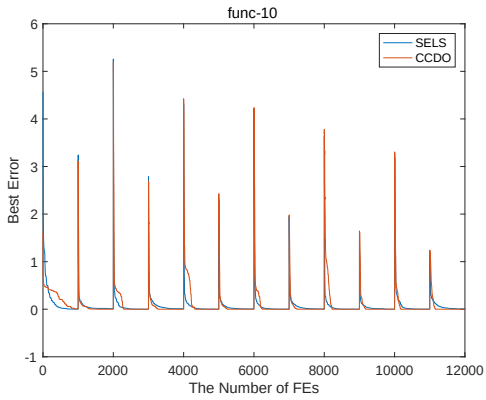
(b) G24-2

(c) G24-3

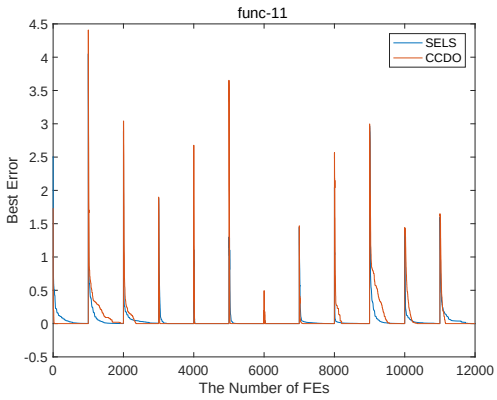
Figure 5.2: Evolutionary curves of SELS and the CCDO method on the first 3 functions



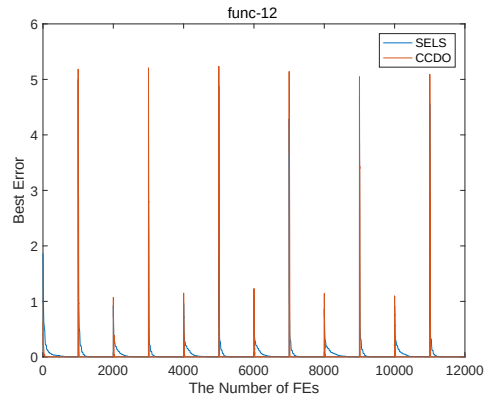
(a) G24-3b



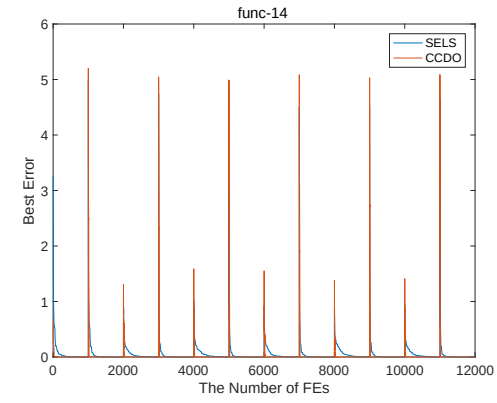
(b) G24-4



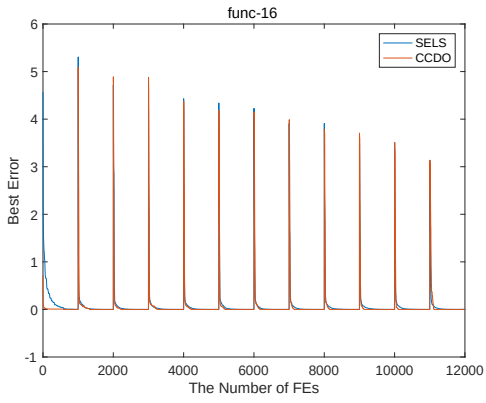
(c) G24-5



(d) G24-6a



(e) G24-6c



(f) G24-7

Figure 5.3: Evolutionary curves of SELS and the CCDO on the last 6 test functions

It can be seen from Table 5.7 that the CCDO method achieved better results than SELS except for only two test functions, G24-4 (dF,dC), G24-5 (dF,dC). The evolutionary curves in Figs. 5.2 and 5.3 explain that the better results achieved on these 7 test functions are attributed to that the CCDO can react quickly once a change is detected. The CCDO method did not perform so well on test functions G24-4 (dF,dC) and G24-5 (dF,dC) because the solution sets found by the competitive co-evolutionary process on them are not good enough. In future work, the performance can be further enhanced by an improved co-evolutionary search process. Moreover, random individuals can be introduced for intialisation in case that the solution set found is not good enough.

Comparison with LTFR-DSPSO

Table 5.8 summarises the mean and standard deviation of the modified offline error over 50 runs obtained by the proposed CCDO+Repair and LTFR-DSPSO under the change frequency of 1000 FEs. In Table 5.8, the better result obtained on each test function is marked in bold. It can be seen from Table 5.8 that the CCDO+Repair achieved better results than LTFR-DSPSO except for only three test functions, G24-4 (dF,dC), G24-6a (dF,fC) and G24-6c (dF,fC).

Table 5.8: Comparison results between the LTFR-DSPSO method and the CCDO+Repair method under the change frequency of 1000 FEs. The better result is marked in **bold**.

Func	LTFR-DSPSO	CCDO+Repair
G24-l (dF,fC)	6.09e−06±4.24e−05	**7.09e−08±1.96e−07**
G24-2 (dF,fC)	8.54e−04±3.38e−03	**6.15e−04±3.07e−03**
G24-3 (fF,dC)	3.64e−05±1.13e−04	**7.87e−11±3.04e−10**
G24-3b (dF,dC)	3.82e−05±1.17e−04	**7.80e−08±3.78e−07**
G24-4 (dF,dC)	**5.45e−06±3.79e−05**	1.81e−02±9.07e−02
G24-5 (dF,dC)	7.00e−05±4.89e−04	**9.92e−07±4.94e−06**
G24-6a (dF,fC)	**2.21e−18±9.28e−18**	5.91e−04±9.80e−04
G24-6c (dF,fC)	**1.91e−18±8.82e−18**	6.39e−04±1.35e−03
G24-7 (fF,dC)	5.11e−06±3.55e−05	**1.28e−13±1.17e−13**

Comparison Results under Change Frequency of 500 FEs and 100 FEs

To test whether CCDO can show a fast adaptation to fast changes, experiments were conducted under two other change frequencies in addtion to 1000 FEs to investigate the effect on the performance of compared algorithms. In the experiments, the change frequency was set to 100 FEs and 500 FEs, respectively. The frequency of 100 FEs represents that the change happens very fast. The proposed CCDO method was tested on the 9 test functions under each change frequency. As no results were found for LTFR-DSPSO under the change frequency of 100 FEs, CCDO was compared only to SELS in this experiment. Tables 5.9 and 5.10 present the comparison results between SELS and CCDO under change frequencies of 500 FEs and 100 FEs, respectively.

Table 5.9: Comparison results between SELS and the CCDO method under the change frequency of 500 FEs. The better result obtained on each test function is marked in **bold**.

Func	SELS	CCDO
G24-l (dF,fC)	6.18e−02±1.65e−02	**2.42e−02±4.11e−02**
G24-2 (dF,fC)	8.98e−02±1.48e−02	**3.14e−02±1.48e−02**
G24-3 (fF,dC)	8.75e−02±2.94e−02	**3.00e−02±1.67e−02**
G24-3b (dF,dC)	1.12e−01±2.86e−02	**9.73e−02±7.24e−02**
G24-4 (dF,dC)	**1.52e−01±3.70e−02**	2.34e−01±9.20e−02
G24-5 (dF,dC)	**9.90e−02±1.54e−02**	1.51e−01±4.08e−02
G24-6a (dF,fC)	1.08e−01±1.45e−02	**6.54e−02±1.30e−02**
G24-6c (dF,fC)	1.09e−01±1.74e−02	**6.28e−02±1.43e−02**
G24-7 (fF,dC)	**1.36e−01±2.02e−02**	1.48e−01±2.68e−02

It can be seen by comparing Table 5.7 and Table 5.9 that the advantage of CCDO over SELS does not change too much when the change frequency changes from 1000 FEs to 500 FEs. However, when the change frequency changes from 1000 FEs or 500 FEs to 100 FEs, it can be seen by comparing Table 5.7 or Table 5.9 and Table 5.10 that the advantage of CCDO over SELS becomes more obvious. For all 9 test functions, CCDO obtained better results than SELS. The comparison between Table 5.7 or Table 5.9 and Table 5.10 can demonstrate the potential advantage of CCDO, that is, a fast adaptation to fast changes. As none of the existing algorithms can peform well on DCOPs with very

fast changes, CCDO offers a direction to optimise such problems.

Table 5.10: Comparison results between SELS and the CCDO method under the change frequency of 100 FEs. The better result obtained on each test function is marked in **bold**.

Func	SELS	CCDO
G24-l (dF,fC)	4.05e−01±9.64e−02	**1.10e−01±1.08e−01**
G24-2 (dF,fC)	3.96e−01±5.56e−02	**1.42e−01±4.23e−02**
G24-3 (fF,dC)	4.87e−01±1.35e−01	**1.51e−01±6.97e−02**
G24-3b (dF,dC)	5.43e−01±8.92e−02	**4.91e−01±1.72e−01**
G24-4 (dF,dC)	8.54e−01±1.29e−01	**7.49e−01±1.23e−01**
G24-5 (dF,dC)	5.25e−01±8.99e−02	**4.73e−01±1.12e−01**
G24-6a (dF,fC)	5.53e−01±9.16e−02	**2.64e−01±9.16e−02**
G24-6c (dF,fC)	5.10e−01±7.52e−02	**3.05e−01±9.40e−02**
G24-7 (fF,dC)	9.43e−01±2.05e−01	**6.43e−01±1.10e−01**

5.4 Chapter Summary

This chapter aimed to investigate whether identifying a set of promising solutions in advance would be beneficial to DCOPs and how to achieve such solutions. To answer these two questions, we proposed a new dynamic constrained optimisation approach, CCDO, which dynamically maintains a solution set that are obtained by the use of ComC at the beginning, and conducts online local search by using this set as initial solutions once a change has been detected. During the online optimisation process, the solution set is updated by the best solutions obtained on encountered environments.

To verify the efficacy of the proposed CCDO method, 9 DCOP benchmark test functions were used and two state-of-the-art optimisation methods, SELS and LTFR-DSPSO, were used for comparison. Experimental studies have shown that the CCDO method generally performed better than SELS and LTFR-DSPSO on the 9 test functions. This demonstrates that identifying a set of promising solutions in advance is more beneficial to DCOPs than searching with randomly generated initial solutions. We also conducted further experiments to check the effectiveness of the environmental evolution in the co-

evolutionary search process. The experimental results demonstrated that the evolution of environments is more suitable for problems in which the smallest ideal solution set has an unlimited size compared to the random method with fixed environments.

In the field of dynamic optimisation, DOPs with very fast changes are the most challenging problems. When the changes happen very fast, the existing diversity-driven approaches, memory approaches and prediction approaches can not work well. The diversity-driven approaches do not have enough time to find a satisfying solution. As the time for optimisation is generally very short, the memory approaches can not have optimal solutions to archive and the prediction approaches can not have useful samples for accurate prediction. However, the proposed CCDO offers a way to address fast-changing DCOPs by identifying a set of promising solutions beforehand. Identifying a set of promising solutions beforehand can shorten the time needed to find a satisfying solution under each change. The experimental results under different change frequencies also demonstrated that CCDO performed well especially under very fast changes.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

The thesis at least partially answers the research questions in Sections 1.3.1-1.3.3. This chapter summarises the contributions of the thesis and gives directions for future work related to the work presented in this thesis.

6.1 Contributions

This thesis is dedicated to develop specific and efficient multi-species EAs with good decomposition and collaboration strategies to address different complex optimisation problems. The main contributions of the thesis are as follows.

6.1.1 Novel Collaboration Strategies When Using CooC to Solve CE Problems

In Chapter 3, two novel collaboration strategies were developed when using cooperative co-evolution (CooC) to address two typical types of CE problems. The first CE problem is the design of a product that has several different parts, and the other is the quasi-separable multidisciplinary design optimisation (MDO) problem in which different disciplines share some design variables. For both design problems, the decomposition is assumed to be known beforehand. The first CE problem is decomposed based on the product structure and the second CE problem is decomposed based on the discipline.

For the first problem, a new novelty-driven CooC algorithm, NDCC-SS, was proposed to evolve the decomposed sub-problems. In NDCC-SS, a computationally efficient novelty calculation is used and a stochastic selection strategy with an adaptive probability are employed to decide whether evaluating individuals based on their fitness or novelty. The efficacy of the proposed novelty-driven CooC algorithm was studied on the single universal electric motor (UEM) design problem. It was shown by experiments that NDCC-SS helped obtain designs of higher quality compared to normal CooC and other novelty-driven CooC methods.

For the second problem, a novel CooC-based concurrent design method, CCDM, was proposed to evolve the decomposed sub-problems. The CCDM method handles the shared variables through the use of duplicates and consistency constraints, in which the stochastic ranking method with an adaptive probability was employed to deal with the constraints. It achieved designs of higher quality in comparison to other MDO methods on a general MDO problem and the design of multiple UEMs that have common design variables.

Additionally, Chapter 3 systematically investigated how the communication frequency among sub-populations affects the performance of NDCC-SS and CCDM. By using different problems and communication costs, the experimental results showed that the best frequency varies as the problem and communication cost change. Motivated by this, a self-adaptive method, CFS, was developed to adjust a good communication frequency automatically during the concurrent design process. The experimental studies showed that CFS is a very competitive communication strategy for both NDCC-SS and CCDM.

6.1.2 A New Dynamic Handling Technique based on Speciation Methods for DCOPs

In Chapter 4, a novel speciation-based method, called speciated evolution with local search (SELS), was proposed to address DCOPs by locating and tracking feasible regions. DCOPs have specific characteristics like the switch of global optima between disconnected feasible regions that do not exist in dynamic optimisation problems with bounded

constraints or without constraints. This poses difficulties for some existing dynamic optimisation strategies.

To address DCOPs, SELS uses speciation methods to make comparisons and does crossover between similar individuals. Through doing this, individuals can automatically search different fesible regions in the search space, and good solutions can be maintained in different feasible regions. This can help SELS react quickly when the global optimal solution switches to another feasible region. Additionally, SELS uses a local search strategy to promote exploitation of the promising regions to quickly find a new optimum. The experimental studies in Chapter 4 demonstrated that SELS generally reacts faster to environmental changes when compared to the state-of-the-art algorithms on a benchmark set of DCOPs.

6.1.3 A New Dynamic Handling Technique based on ComC for Fast-Changing DCOPs

In the field of dynamic optimisation, DOPs with very fast changes are the most challenging problems. When the changes happen very fast, the existing diversity-driven approaches, memory approaches and prediction approaches can not work well. The diversity-driven approaches do not have enough time to find a satisfactory solution. As the time for optimisation is generally very short, the memory approaches can not have optimal solutions to archive and thus the prediction approaches can not have useful samples for accurate prediction.

Considering these, in Chapter 5, a novel dynamic handling strategy for fast-changing DCOPs, CCDO, was proposed, which uses competitive co-evolution (ComC) to find a set of good solutions beforehand and then uses the solution set for initialisation once an environmental change happens. In CCDO, a DCOP is first decomposed into the solution space and environment space. Then, the ComC method is applied to co-evolve them to find a set of promising solutions for all environments. The CCDO method offers a way to address fast-changing DCOPs by identifying a set of promising solutions beforehand,

which can shorten the time needed to find a satisfactory solution under each change. The experimental results under different change frequencies demonstrated that CCDO reacts faster than other methods for DCOPs especially under very fast changes.

6.2 Future Work

In this section, we would like to point out some possible research directions in the future.

- The work in Chapter 3 only considered UEM design problems and a general MDO problem. In future work, the two proposed CooC-based concurrent optimisation methods will be tested on more benchmark and real-world problems. The experimental studies in Chapter 3 only considered single objective optimisation. In reality, many CE problems are multi-objective design optimisation problems. When developing methods for multi-objective design problems, a different novelty calculation method and common variable handling technique might be required. This can be another direction of future work.

- The work in Chapter 4 only tested the proposed SELS method on 11 DCOP benchmark functions. In future work, the real-world DCOP problems will be considered. In the proposed SELS method, the choice and the length of local search strategy as well as on which individual to do local search can have a big effect on the peformance of SELS. Future studies will be conducted along this direction. Moreover, other multi-species EAs can also be applied to better address DCOPs. The Euclidean distance might not work when the number of decision parameters increases, other distance metrics that work well in a high-dimensional space should be applied or developed.

- The work in Chapter 5 is only the first step to validate whether the proposed CCDO approach is able to react quickly to environmental changes. In future work, the CCDO method will be tested on more dynamic test problems. Note that the

CCDO method can be generally applied to dynamic optimisation problems (dynamic constraint satisfaction problems, dynamic optimisation without constraints or with bounded constraints). As the cost in co-evolutionary set search process was ignored in the experiments, in future work, the performance of CCDO will be evaluated with the co-evolutionary search cost being counted in. Moreover, the competitive co-evolutionary search process will be improved to have a better coverage. It should be noted that every component of the CCDO method (SP evaluation, EP evaluation, re-intialisation, local search, online update, etc.) can be replaced by other better mechanisms.

- Some open issues also arise from Chapter 5. First, for a problem in which the smallest ideal solution set has an unlimited size, whether there exists a limited solution set on which conducting local search can find the optimal solution for the sub-problem under any environment needs some theoretical investigation. Second, the issue of a suitable set size in the proposed dynamic optimisation framework needs to be studied. Third, finding a set of good solutions for a class of sub-problems is similar to a set covering problem. For such a problem, whether and when ComC is the most effective method needs further investigation.

APPENDIX

THE BASELINE UEM DESIGN PROBLEM FORMULATION

This appendix gives the details in calculating $H(\boldsymbol{x})$, $Mass(\boldsymbol{x})$, $\eta(\boldsymbol{x})$, $P(\boldsymbol{x})$, $T(\boldsymbol{x})$ and the penalty with respect to the 8 design parameters, $\{N_c, N_s, A_{wf}, A_{wa}, I, r_o, t, L\}$ [117]. Table 1 gives the values of some constants used in the calculation.

Table 1: Values of constants

Constant	Value	Constant	Value
V	115[V]	ρ	$1.69 * 10^{-8}[\Omega * m]$
l_{gap}	0.0007[m]	p_{field}	2
ρ_{steel}	$7850[\frac{kg}{m^3}]$	ρ_{copper}	$8960[\frac{kg}{m^3}]$
μ_{air}	1	π	3.14159

Magnetizing Intensity: $H(\boldsymbol{x})$

$$H = \frac{2 * N_s * I}{l_c + l_r + 2 * l_{gap}} \tag{1}$$

where:

$$l_c = \pi * \frac{(2 * r_o + t)}{2}$$

$$l_r = 2 * (r_o - t - l_{gap})$$

Weight: $Mass(\boldsymbol{x})$

$$Mass = M_{stator} + M_{armature} + M_{windings} \tag{2}$$

where:

$$M_{stator} = \pi * (r_o^2 - (r_o - t)^2) * L * \rho_{steel}$$

$$M_{armature} = \pi * (r_o - t - l_{gap})^2 * L * \rho_{steel}$$

$$M_{windings} = ((2 * L + 4 * (r_o - t - l_{gap})) * N_c * A_{wa}$$

$$+ (2 * L + 4 * (r_o - t)) * 2 * N_s * A_{wf}) * \rho_{copper}$$

Power: $P(\boldsymbol{x})$

$$P = P_{in} - P_{losses} = V * I - P_{losses} \tag{3}$$

where:

$$P_{losses} = P_{brush} + P_{copper}$$

$$P_{brush} = 2 * I$$

$$P_{copper} = I^2 * (R_a + R_s)$$

$$R_a = \frac{\rho * (2 * L + 4 * (r_o - t - l_{gap})) * N_c}{A_{wa}}$$

$$R_s = \frac{\rho * p_{field} * (2 * L + 4 * (r_o - t)) * N_s}{A_{wf}}$$

Efficiency: $\eta(\boldsymbol{x})$

$$\eta = \frac{P}{P_{in}} = \frac{P}{V * I} \tag{4}$$

Torque: $T(\boldsymbol{x})$

$$T = K * \phi * I = \frac{N_c}{\pi} * \phi * I \tag{5}$$

where:

$$\phi = \frac{S}{R}$$
$$S = N_s * I$$
$$R = R_s + R_r + 2 * R_a$$
$$R_s = \frac{l_c}{2 * \mu_{steel} * \mu_0 * t * L}$$
$$R_r = \frac{l_r}{\mu_{steel} * \mu_0 * l_r * L}$$
$$R_a = \frac{l_{gap}}{\mu_{air} * \mu_0 * l_r * L}$$

and

$$\mu_{steel} = \begin{cases} -0.22791 * H^2 + 52.411 * H + 3115.8, & H \leq 220 \\ 11633.5 - 1486.33 * ln(H), & 220 < H \leq 1000 \\ 1000, & H > 1000 \end{cases}$$

Penalty

The penalty is calculated as the sum of the penalty with respect to each constraint in Eq. 3.8. Details are given as follows:

$$penalty = p_H + p_{rt} + p_M + p_\eta + p_P + p_T \tag{6}$$

where:

$$p_H = \begin{cases} 0, & \text{if } H \leq 5000[Amp * turns/m] \\ 1.0 + ((H - 5000) * 0.01)^2, & \text{otherwise} \end{cases}$$

$$p_{rt} = \begin{cases} 0, & \text{if } r_o \geq t \\ 1.0 + (t - r_o)^2, & \text{otherwise} \end{cases}$$

$$p_M = \begin{cases} 0, & Mass \leq 2[kg] \\ 1.0 + ((Mass - 2) * 100)^2, & \text{otherwise} \end{cases}$$

$$p_\eta = \begin{cases} 0, & \eta \geq 0.15 \\ 1.0 + ((0.15 - \eta) * 100)^2, & \text{otherwise} \end{cases}$$

$$p_P = \begin{cases} 0, & |P - 300[W]| \leq 5[W] \\ ((|300 - P| - 5) * 0.1)^2, & \text{otherwise} \end{cases}$$

and

$$p_T = \begin{cases} 0, & |T - a| \leq 0.02 * T \\ ((|T - a| - 0.02 * T) * 1000)^2, & \text{otherwise} \end{cases}$$

LIST OF REFERENCES

[1] MATLAB. (2017b). Constrained nonlinear optimization algorithms. *Optimization Toolbox Documentation (online). Available: https://cn.mathworks.com/help/optim/ug/constrained-nonlinear-optimization-algorithms.html*, 2017.

[2] R. Addo-Tenkorang. Concurrent engineering: A review literature report. In *Proceedings of the World Congress on Engineering and Computer Science*, pages 1074–1080. IEEE, 2011.

[3] Satish VK Akundi, Timothy W Simpson, and Patrick M Reed. Multi-objective design optimization for product platform and product family design using genetic algorithms. In *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 999–1008. American Society of Mechanical Engineers, 2005.

[4] Maria-Yaneli Ameca-Alducin, Efrén Mezura-Montes, and Nicandro Cruz-Ramirez. Differential evolution with combined variants for dynamic constrained optimization. In *2014 IEEE Congress on Evolutionary Computation (CEC'14)*, pages 975–982. IEEE, 2014.

[5] 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 Annual Conference on Genetic and Evolutionary Computation (GECCO'15)*, pages 241–248. ACM, 2015.

[6] Luis Miguel Antonio and Carlos A Coello Coello. Use of cooperative coevolution for solving large scale multiobjective optimization problems. In *Proceedings of the 2013 IEEE Congress on Evolutionary Computation (CEC'13)*, pages 2758–2765. IEEE, 2013.

[7] C. Aranha and H. Iba. The memetic tree-based genetic algorithm and its application to portfolio optimization. *Memetic Computing*, 1(2):139–151, 2009.

[8] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[9] Charles-S Beightler, Donald-T Phillips, Ron-S Dembo, G-V Reklaitis, and R-E-D Woolsey. *Applied geometric programming*. John Wiley & Sons, Inc., Distributed by Instrument Society of America, 1975.

[10] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies–a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[11] Paul T Boggs and Jon W Tolle. Sequential quadratic programming. *Acta numerica*, 4:1–51, 1995.

[12] Jürgen Branke, Thomas Kaußler, Christian Smidt, and Hartmut Schmeck. A multi-population approach to dynamic optimization problems. In *Evolutionary Design and Manufacture*, pages 299–307. Springer, 2000.

[13] Chenyang Bu, Wenjian Luo, and Lihua Yue. Continuous dynamic constrained optimization with ensemble of locating and tracking feasible regions strategies. *IEEE Transactions on Evolutionary Computation*, 21(1):14–33, 2017.

[14] Anthony Bucci and Jordan B Pollack. On identifying global optima in cooperative coevolution. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO'05)*, pages 539–544. ACM, 2005.

[15] Mario Campos and Renato Krohling. Bare bones particle swarm with scale mixtures of gaussians for dynamic constrained optimization. In *Proceedings of the 2009 IEEE Congress on Evolutionary Computation (CEC'14)*, pages 202–209. IEEE, 2014.

[16] Mauro Campos and Renato A Krohling. Entropy-based bare bones particle swarm for dynamic constrained optimization. *Knowledge-Based Systems*, 97:203–223, 2016.

[17] Qifeng Chen and Jinhai Dai. Distributed coevolutionary multidisciplinary design optimization: A flexible approach. In *The 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, 4-6 September 2002, Altanta, Georgia*, 2002.

[18] Qifeng Chen and Jinhai Dai. A distributed multi-objective evolutionary algorithm and application in missile design. In *The 2002 International Conference on Control and Automation (ICCA'02)*, pages 197–197. IEEE, 2002.

[19] Siang Y Chong, Mei K Tan, and Jonathon David White. Observing the evolution of neural networks learning to play the game of othello. *IEEE Transactions on Evolutionary Computation*, 9(3):240–251, 2005.

[20] Siang Yew Chong and Xin Yao. Behavioral diversity, choices and noise in the iterated prisoner's dilemma. *IEEE Transactions on Evolutionary Computation*, 9(6):540–551, 2005.

[21] Piya Chootinan and Anthony Chen. Constraint handling in genetic algorithms using a gradient-based repair method. *Computers and Operations Research*, 33(8):2263–2281, 2006.

[22] Helen G Cobb. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical report, DTIC Document, 1990.

[23] A Colorni, M Dorigo, and V Maniezzo. Distributed optimization by ant colonies. In *Proceedings of the 1st European Conference on Artificial Life (ECAL'91)*, pages 134–142, Amsterdam, The Netherlands, 1991. Elsevier.

[24] W J Conover and R L Iman. On multiple-comparisons procedures. informal report. Technical report, Los Alamos Scientific Laboratory., 1979.

[25] Carlos Cruz, Juan R González, and David A Pelta. Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing*, 15(7):1427–1448, 2011.

[26] Giuseppe Cuccu and Faustino Gomez. When novelty is not enough. In *European Conference on the Applications of Evolutionary Computation*, pages 234–243. Springer, 2011.

[27] Paul J Darwen and Xin Yao. Does extra genetic diversity maintain escalation in a co-evolutionary arms race. In *International Journal of Knowledge-Based Intelligent Engineering Systems*, volume 4, pages 191–200. Citeseer, 2000.

[28] Susmita De, Sankar K Pal, and Ashish Ghosh. Genotypic and phenotypic assortative mating in genetic algorithm. *Information Sciences*, 105(1):209–226, 1998.

[29] Kenneth Alan De Jong. *Analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.

[30] Kalyanmoy Deb and Ram B Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9(3):1–15, 1994.

[31] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[32] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[33] Philippe Dépincé, Benoît Guédas, and Jérôme Picard. Multidisciplinary and multiobjective optimization: Comparison of several methods. In *7th World Congress on Structural and Multidisciplinary Optimization*, 2007.

[34] Alex Dmitrienko, Ajit C Tamhane, and Frank Bretz. *Multiple testing problems in pharmaceutical statistics*. CRC press, 2009.

[35] Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science (MHS'95)*, pages 39–43, Piscataway, NJ, USA, 1995. IEEE.

[36] Agoston E Eiben, James E Smith, et al. *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin/Heidelberg, Germany, 2003.

[37] Patryk Filipiak and Piotr Lipinski. Infeasibility driven evolutionary algorithm with feed-forward prediction strategy for dynamic constrained optimization problems. In *Applications of Evolutionary Computation*, pages 817–828. Springer, 2014.

[38] Patryk Filipiak and Piotr Lipinski. Making idea-arima efficient in dynamic constrained optimization problems. In *Applications of Evolutionary Computation*, pages 882–893. Springer, 2015.

[39] H Finner. On a monotonicity problem in step-down multiple test procedures. *Journal of the American Statistical Association*, 88(423):920–923, 1993.

[40] David B Fogel. *Evolutionary Computation: the Fossil Record*. Wiley-IEEE Press, New York, USA, 1998.

[41] David B Fogel. Introduction to evolutionary computation. *Evolutionary Computation*, 1:1–3, 2000.

[42] Lawrence J Fogel, Alvin J Owens, and Michael J Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, Chichester, WS, UK, 1966.

[43] Timo Friedrich and Stefan Menzel. A cascaded evolutionary multi-objective optimization for solving the unbiased universal electric motor family problem. In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC'14)*, pages 3184–3191. IEEE, 2014.

[44] Gianpaolo Ghiani, Francesca Guerriero, Gilbert Laporte, and Roberto Musmanno. Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies. *European Journal of Operational Research*, 151(1):1–11, 2003.

[45] Alessandro Giassi, F Bennis, and J-J Maisonneuve. Multidisciplinary design optimisation and robust design approaches applied to concurrent design. *Structural and Multidisciplinary Optimization*, 28(5):356–371, 2004.

[46] David E Goldberg, Jon Richardson, et al. Genetic algorithms with sharing for multimodal function optimization. In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Hillsdale, NJ: Lawrence Erlbaum, 1987.

[47] Jorge Gomes, Pedro Mariano, and Anders Lyhne Christensen. Avoiding convergence in cooperative coevolution with novelty search. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1149–1156. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[48] Jorge Gomes, Pedro Mariano, and Anders Lyhne Christensen. Devising effective novelty search algorithms: a comprehensive empirical study. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO'15)*, pages 943–950. ACM, 2015.

[49] Jorge Gomes, Pedro Mariano, and Anders Lyhne Christensen. Novelty-driven cooperative coevolution. *Evolutionary computation*, 25(2):275–307, 2017.

[50] John J Grefenstette et al. Genetic algorithms for changing environments. In *Proceedings of the 2nd Conference on Parallel Problem Solving from Nature–PPSN II*, volume 2, pages 137–144, 1992.

[51] S.M.K Hasan, R. Sarker, D. Essam, and D. Cornforth. Memetic algorithms for solving job-shop scheduling problems. *Memetic Computing*, 1(1):69–83, 2009.

[52] W Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D: Nonlinear Phenomena*, 42(1-3):228–234, 1990.

[53] Robert Hinterding, Zbigniew Michalewicz, and Thomas C Peachey. Self-adaptive genetic algorithm for numeric functions. In *Proceedings of the 4th Conference on Parallel Problem Solving from Nature–PPSN IV*, pages 420–429. Springer, 1996.

[54] Pei Yee Ho and Kazuyuki Shimizu. Evolutionary constrained optimization using an addition of ranking method and a percentage-based tolerance value adjustment scheme. *Information Sciences*, 177(14):2985–3004, 2007.

[55] H Holland John. *Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, Ann Arbor, Michigan, USA, 1975.

[56] Colin G Johnson and Juan Jesús Romero Cardalda. Genetic algorithms in visual art and music. *Leonardo*, 35(2):175–184, 2002.

[57] Swaroop Kalasapur, Mohan Kumar, and Behrooz A Shirazi. Dynamic service composition in pervasive computing. *IEEE Transactions on Parallel and Distributed Systems*, 18(7):907–918, 2007.

[58] Namwoo Kang, Michael Kokkolaras, and Panos Y Papalambros. Solving multiobjective optimization problems using quasi-separable mdo formulations and analytical target cascading. *Structural and Multidisciplinary Optimization*, 50(5):849–859, 2014.

[59] Aida Khajavirad, Jeremy J Michalek, and Timothy W Simpson. An efficient decomposed multiobjective genetic algorithm for solving the joint product platform selection and product family design problem with generalized commonality. *Structural and Multidisciplinary Optimization*, 39(2):187–201, 2009.

[60] Hyung Min Kim, Nestor F Michelena, Panos Y Papalambros, and Tao Jiang. Target cascading in optimal system design. *Journal of Mechanical Design*, 125(3):474–480, 2003.

[61] John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, 1994.

[62] Viswanathan Krishnan, Steven D Eppinger, and Daniel E Whitney. A model-based framework to overlap product development activities. *Management Science*, 43(4):437–451, 1997.

[63] Andrew Kusiak. *Concurrent engineering: automation, tools, and techniques*. John Wiley & Sons, 1993.

[64] Andrew Kusiak and Nick Larson. Decomposition and representation methods in mechanical design. *Journal of Vibration and Acoustics*, 117(B):17–24, 1995.

[65] Eric-Wubbo Lameijer, Thomas Bäck, Joost N Kok, and Ad P Ijzerman. Evolutionary algorithms in drug design. *Natural Computing*, 4(3):177–243, 2005.

[66] Joel Lehman and Kenneth O Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *Artificial Life*, pages 329–336, 2008.

[67] Joel Lehman and Kenneth O Stanley. Efficiently evolving programs through the search for novelty. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO'10)*, pages 837–844. ACM, 2010.

[68] Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011.

[69] Changhe Li, Trung Thanh Nguyen, Ming Yang, Shengxiang Yang, and Sanyou Zeng. Multi-population methods in unconstrained continuous dynamic environments: The challenges. *Information Sciences*, 296:95–118, 2015.

[70] Jian-Ping Li, Marton E Balazs, Geoffrey T Parks, and P John Clarkson. A species conserving genetic algorithm for multimodal function optimization. *Evolutionary Computation*, 10(3):207–234, 2002.

[71] Xiaodong Li. Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO'04)*, pages 105–116. Springer, 2004.

[72] Xiaodong Li, Michael G Epitropakis, Kalyanmoy Deb, and Andries Engelbrecht. Seeking multiple solutions: an updated survey on niching methods and their applications. *IEEE Transactions on Evolutionary Computation*, 21(4):518–538, 2017.

[73] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. Constrained novelty search: A study on game content generation. *Evolutionary Computation*, 23(1):101–129, 2015.

[74] Jun Lin, Yanjun Qian, Wentian Cui, and Zhanli Miao. Overlapping and communication policies in product development. *European Journal of Operational Research*, 201(3):737–750, 2010.

[75] Christoph H Loch and Christian Terwiesch. Communication and uncertainty in concurrent engineering. *Management Science*, 44(8):1032–1048, 1998.

[76] Xiaofen Lu, Stefan Menzel, Ke Tang, and Xin Yao. The performance effects of interaction frequency in parallel cooperative coevolution. In *Simulated Evolution and Learning*, pages 82–93. Springer, 2014.

[77] Xiaofen Lu, Ke Tang, Bernhard Sendhoff, and Xin Yao. A review of concurrent optimisation methods. *International Journal of Bio-Inspired Computation*, 6(1):22–31, 2014.

[78] Xiaofen Lu, Ke Tang, and Xin Yao. Speciated evolutionary algorithm for dynamic constrained optimisation. In *International Conference on Parallel Problem Solving from Nature (PPSN XIV)*, pages 203–213. Springer, 2016.

[79] Sedigheh Mahdavi, Shahryar Rahnamayan, and Mohammad Ebrahim Shiri. Incremental cooperative coevolution for large-scale global optimization. *Soft Computing*, 22(6):2045–2064, 2018.

[80] Samir W Mahfoud. Niching methods for genetic algorithms. *Urbana*, 51(95001):62–94, 1995.

[81] Rammohan Mallipeddi and Ponnuthurai N Suganthan. Ensemble of constraint handling techniques. *IEEE Transactions on Evolutionary Computation*, 14(4):561–579, 2010.

[82] Joaquim RRA Martins and Andrew B Lambe. Multidisciplinary design optimization: a survey of architectures. *American Institute of Aeronautics and Astronautics (AIAA) Journal*, 51(9):2049–2075, 2013.

[83] Yi Mei, Xiaodong Li, and Xin Yao. Cooperative co-evolution with route distance grouping for large-scale capacitated arc routing problems. *IEEE Transactions on Evolutionary Computation*, 18(3):435–449, 2014.

[84] Ole J Mengshoel and David E Goldberg. Probabilistic crowding: Deterministic crowding with probabilistic replacement. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO'99)*, pages 409–416, 1999.

[85] Efrén Mezura-Montes, Carlos A Coello Coello, and Edy I Tun-Morales. Simple feasibility rules and differential evolution for constrained optimization. In *MICAI 2004: Advances in Artificial Intelligence*, pages 707–716. Springer, 2004.

[86] Brad L Miller and Michael J Shaw. Genetic algorithms with dynamic niche sharing for multimodal function optimization. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 786–791. IEEE, 1996.

[87] Angel Kuri Morales and Carlos Villegas Quezada. A universal eclectic genetic algorithm for constrained optimization. In *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing*, volume 1, pages 518–522, 1998.

[88] Jean-Baptiste Mouret. Novelty-based multiobjectivization. In *New Horizons in Evolutionary Robotics*, pages 139–154. Springer, 2011.

[89] F Moussouni, S Kreuawan, S Brisset, F Gillon, P Brochet, and L Nicod. Analytical target cascading for optimal design of railway traction system. In *International Conference on Engineering Optimization*, pages 1–10, 2008.

[90] Prasanth B Nair and Andy J Keane. Coevolutionary genetic adaptation–a new paradigm for distributed multidisciplinary design optimization. In *The 40th AIAA/ASME/ASCE/AHS/ASC Structures Structural Dynamics and Materials Conf., St. Louis, MO, AIAA Paper*, pages 1428–1437. Citeseer, 1999.

[91] Prasanth B Nair and Andy J Keane. Coevolutionary architecture for distributed optimization of complex coupled systems. *American Institute of Aeronautics and Astronautics (AIAA) Journal*, 40(7):1434–1443, 2002.

[92] Trung Thanh Nguyen. *Continuous dynamic optimisation using evolutionary algorithms*. PhD thesis, University of Birmingham, 2011.

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

[94] Trung Thanh Nguyen and Xin Yao. Continuous dynamic constrained optimization–the challenges. *IEEE Transactions on Evolutionary Computation*, 16(6):769–786, 2012.

[95] Mohammad Nabi Omidvar, Xiaodong Li, Zhenyu Yang, and Xin Yao. Cooperative co-evolution for large scale optimization through more frequent random grouping. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC'10)*, pages 1–8. IEEE, 2010.

[96] Kunal Pal, Chiranjib Saha, and Swagatam Das. Differential evolution and offspring repair method based dynamic constrained optimization. In *Swarm, Evolutionary, and Memetic Computing*, pages 298–309. Springer, 2013.

[97] Kunal Pal, Chiranjib Saha, Swagatam Das, Carlos Coello, et al. Dynamic constrained optimization with offspring repair based gravitational search algorithm. In *Proceedings of the 2009 IEEE Congress on Evolutionary Computation (CEC'13)*, pages 2414–2421. IEEE, 2013.

[98] Liviu Panait. Theoretical convergence guarantees for cooperative coevolutionary algorithms. *Evolutionary Computation*, 18(4):581–615, 2010.

[99] Liviu Panait, R Paul Wiegand, and Sean Luke. A visual demonstration of convergence properties of cooperative coevolution. In *International Conference on Parallel Problem Solving from Nature*, pages 892–901. Springer, 2004.

[100] Daniel Parrott and Xiaodong Li. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Transactions on Evolutionary Computation*, 10(4):440–458, 2006.

[101] Thorsten Pohlert. The pairwise multiple comparison of mean ranks package (pmcmr). *R package*, pages 2004–2006, 2014.

[102] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, 2007.

[103] Elena Popovici and Kenneth De Jong. The effects of interaction frequency on the optimization performance of cooperative coevolution. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*, pages 353–360. ACM, 2006.

[104] Mitchell A Potter and Kenneth A De Jong. A cooperative coevolutionary approach to function optimization. In *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature–PPSN III*, pages 249–257. Springer, 1994.

[105] Mitchell A Potter and Kenneth A De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[106] Sébastien Rabeau, Philippe Dépincé, and Fouad Bennis. Collaborative optimization of complex systems: a multidisciplinary approach. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 1(4):209–218, 2007.

[107] Hendrik Richter. Memory design for constrained dynamic optimization problems. In *Applications of Evolutionary Computation*, pages 552–561. Springer, 2010.

[108] Thomas A Roemer, Reza Ahmadi, and Robert H Wang. Time-cost trade-offs in overlapped product development. *Operations Research*, 48(6):858–865, 2000.

[109] Christopher D Rosin and Richard K Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1997.

[110] Kamel Rouibah and Kevin R Caskey. Change management in concurrent engineering from a parameter perspective. *Computers in industry*, 50(1):15–34, 2003.

[111] Thomas P. Runarsson and Xin Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, 2000.

[112] Sancho Salcedo-Sanz. A survey of repair methods used as constraint handling techniques in evolutionary algorithms. *Computer Science Review*, 3(3):175–192, 2009.

[113] Bruno Sareni and Laurent Krahenbuhl. Fitness sharing and niching methods revisited. *IEEE Transactions on Evolutionary Computation*, 2(3):97–106, 1998.

[114] Friedrich Schmid and Rafael Schmidt. Multivariate conditional versions of spearman's rho and related measures of tail dependence. *Journal of Multivariate Analysis*, 98(6):1123–1140, 2007.

[115] Dan Simon. Biogeography-based optimization. *IEEE Transactions on Evolutionary Computation*, 12(6):702–713, 2008.

[116] Timothy W Simpson. Product platform design and customization: status and promise. *AI EDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 18(01):3–20, 2004.

[117] Timothy W Simpson, Jonathan R Maier, and Farrokh Mistree. Product platform design: method and application. *Research in Engineering Design*, 13(1):2–22, 2001.

[118] Gulshan Singh and Kalyanmoy Deb Dr. Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pages 1305–1312. ACM, 2006.

[119] C Stoen. Various collaborator selection pressures for cooperative coevolution for classification. In *International Conference of Artificial Intelligence and Digital Communications (AIDC)*, 2006.

[120] Rainer Storn and Kenneth Price. Differential evolutiona simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, Berkeley, CA, USA, 1995.

[121] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[122] Vladimir Tosic, David Mennie, and Bernard Pagurek. Dynamic service composition and its applicability to e-business software systems–the icaris experience. *Object-Oriented Business Solutions ECOOP*, 2000.

[123] Simon Tosserams, LF Pascal Etman, and JE Rooda. An augmented lagrangian decomposition method for quasi-separable problems in mdo. *Structural and Multidisciplinary Optimization*, 34(3):211–227, 2007.

[124] Nikos Tzevelekos, Michael Kokkolaras, Panos Y Papalambros, Martijn F Hulshof, LF Pascal Etman, and JE Rooda. An empirical local convergence study of alternative coordination schemes in analytical target cascading. In *Proceedings of the 5th World Congress on Structural and Multidisciplinary Optimization*, pages 19–23, 2003.

[125] Hans-Michael Voigt and Jan Matti Lange. Local evolutionary search enhancement by random memorizing. In *Proceedings of the 1998 IEEE International Conference on Computational Intelligence*, pages 547–552. IEEE, 1998.

[126] Karen Willcox and Sean Wakayama. Simultaneous optimization of a multiple-aircraft family. *Journal of Aircraft*, 40(4):616–622, 2003.

[127] Yonggang Xing and Shuo Tang. A distributed coevolutionary multidisciplinary design optimization algorithm. In *The 2010 International Joint Conference on Computational Science and Optimization (CSO)*, volume 2, pages 77–80. IEEE, 2010.

[128] Peng Yang, Ke Tang, and Xiaofen Lu. Improving estimation of distribution algorithm on multimodal problems by detecting promising areas. *IEEE Transactions on Cybernetics*, 45(8):1438–1449, 2015.

[129] Zhenyu Yang, Ke Tang, and Xin Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2985–2999, 2008.

[130] Ali Yassine and Dan Braha. Complex concurrent engineering and the design structure matrix method. *Concurrent Engineering*, 11(3):165–176, 2003.

[131] Ali A Yassine, Ramavarapu S Sreenivas, and Jian Zhu. Managing the exchange of information in product development. *European Journal of Operational Research*, 184(1):311–326, 2008.

[132] Parviz M Zadeh, Vassili V Toropov, and Alastair S Wood. Metamodel-based collaborative optimization framework. *Structural and Multidisciplinary Optimization*, 38(2):103–115, 2009.

[133] Xiaoling Zhang, Hong-Zhong Huang, Zhonglai Wang, Yu Liu, and Yanfeng Li. A pareto set coordination method for analytical target cascading. *Concurrent Engineering*, 21(4):286–295, 2013.

[134] Albert Y. Zomaya and Yee-Hwei Teh. Observations on using genetic algorithms for dynamic load-balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(9):899–911, 2001.